1-8-2020

A Simulation Environment with Reduced Reality Gap for Testing Autonomous Vehicles

Kaival Kamleshkumar Patel
University of Windsor

A Simulation Environment with Reduced Reality Gap for Testing Autonomous Vehicles

By

Kaival Kamleshkumar Patel

A THESIS

Submitted to the Faculty of Graduate Studies

Through Computer Science

In Partial Fulfilment of the Requirements for

The Degree of Master of Science at the

University of Windsor

Windsor, Ontario, Canada

2020

A Simulation Environment with Reduced Reality Gap for Testing Autonomous Vehicles

By

Kaival Kamleshkumar Patel

Approved by:

_____

M. Monfared

Department of Mathematics and Statistics

_____

D. Wu

School of Computer Science

_____

X. Yuan, Advisor

School of Computer Science

17th January, 2020

Declaration of Originality

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office and that this thesis has not been submitted for a higher degree to any other University or Institution.

Abstract

In order to facilitate acceptance and ensure safety, autonomous vehicles must be tested not only in typical and relatively safe scenarios but also in dangerous and less frequent scenarios. Recent pedestrian fatalities caused by test vehicles of the front-running giants like Google and Tesla suffice the fact that Autonomous Vehicle technology is not yet mature enough and still needs rigorous exposure to a wide range of traffic, landscape, and natural conditions on which the Autonomous Vehicles can be trained on to perform as expected in real traffic conditions. Simulation Environments have been considered as an efficient, safe, flexible and cost-effective option for the training, testing, and validation of Autonomous Vehicle technology. While ad-hoc task-specific use of simulation in Autonomous Driving research is widespread, simulation platforms that bridge the gap between simulation and reality are limited. This research proposes to set up a highly realistic simulation environment (using CARLA driving simulator) to generate realistic data to be used for Autonomous Driving research. Our system is able to recreate the original traffic scenarios based on prior information about the traffic scene. Furthermore, the system will allow to make changes to the original scenarios and create various desired testing scenarios by varying the parameters of traffic actors, such as location, trajectory, speed, motion states, etc. and hence collect more data with ease.

Dedication

I would like to dedicate this work to the God Almighty and my entire family, including my parents, *Mrs. Dipikaben* and *Mr. Kamleshkumar*, my grandparents, *Mrs. Vidyaben*, and, *Mr. Shantilal*, aunt-uncle *Mrs. Smitaben* and *Mr. Dipteshkumar*, and my cousin *Mr. Achal Patel*. I am grateful for their unconditional love and continuous support in every aspect of my life. Also, I want to use this opportunity to express gratitude to my maternal uncle, *Late Mr. Sharadkumar Patel*, for his encouragement and support for my Master's venture.

Acknowledgment

Firstly, I would like to express my sincere gratitude to my supervisor, Dr. Xiaobu Yuan, who has supported and encouraged me throughout my Master's program with his awareness, knowledge, and expertise in this field of research. I feel privileged to have him as my supervisor, as he actively guided me through every aspect of my research work.

I would also like to acknowledge the constructive feedback from Dr. Dan Wu and Dr. Mehdi Monfared, that helped me to improve my research work.

I want to thank my family, true friends and all other well-wishers for their immense love, support, and motivation.

Table of Contents

Table of Tables

Table of Figures

Abbreviations

AI: Artificial Intelligence

GM: General Motors

Lidar: Light Detection and Ranging

AV: Autonomous Vehicle

GPS: Global Positioning System

IMU: Inertial Measurement Unit

PID: Proportional Integral Derivative

MPC: Model Predictive Control

AVS: Autonomous Visualization System

R-CNN: Region-based Convolutional Neural Network

RPN: Region Proposal Network

VGG: Visual Geometry Group

ROI: Region of Interest

YOLO: You Look Only Once

MS COCO: Microsoft Common Objects in Context

SYNTHIA: Synthetic collection of Imagery and Annotations

VR: Virtual Reality

PPU: Physics Processing Unit

API: Application Program Interface

GPU: Graphics Processing Unit

UE4: Unreal Engine 4

HITL: Hardware in the loop

GPL: General Public License

GAN: Generative Adversarial Network

DR: Domain Randomization

VGI: Volunteered Geographic Information

IoT: Internet of Things

DSA: Dynamic Spatial Attention

RNN: Recurrent Neural Network

TTC: Time to Collision

Chapter 1: Introduction

1.1 Overview

A self-driving car, also known as an autonomous car or driverless car, is a vehicle that uses a combination of sensors, cameras, radars, and artificial intelligence (AI), to travel between destinations without the need of any human effort. In the past five years, autonomous driving has gone from “maybe possible” to “now commercially available” and hence become a concrete reality. Autonomous driving may pave the way for future systems where computers take over the art of driving. The global self-driving car market is segmented based on the type of vehicle, product type, application of the car, technology components(hardware and software), and geography [1]. Automotive Innovators like Waymo and Tesla have been leading the self-driving car industry for long. On the other hand, legacy companies like GM, Ford, Toyota, Nissan, etc. have more recently joined the chase pumping billions of dollars into the research and development of autonomous vehicles.

1.2 Need for self-driving cars

Traffic fatalities claim more than a million lives a year around the world. This public safety crisis that is mainly caused by driver error has largely been overlooked and considered a necessary trade-off in our commuting economy. But with the development of advanced sensor technologies (e.g. cameras, radars, Lidars, etc.), and perception systems guided by big data, artificial intelligence, and increasing processing power, we are fast approaching the day when self-driving vehicles can do a better job than human drivers [2]. Having self-driving cars could also give people a lot more free time. Commutes might be spent working on projects, talking to other passengers and doing many other productive tasks. Fully autonomous cars will be able to drive together in perfect

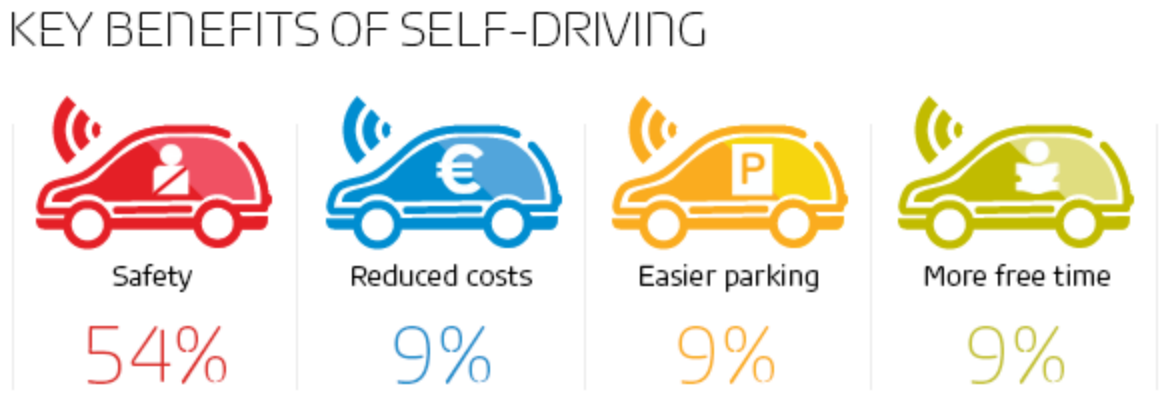

Figure 1: Benefits of self-driving vehicles [3]

harmony like a swarm of bees, radically reducing the traffic jams and increasing the traffic flow. It is also projected that the advent of autonomous vehicles will cut down the insurance premiums, accident-related costs, driving-related fines and, increase fuel efficiency.

1.3 Architecture of Self-Driving Cars

The architecture of self-driving vehicles comprises of four main subsystems: Sensors, Perception, Planning and Control. These subsystems act together to perceive the environment around the autonomous vehicle, detect the paths, plan a route to the destination, predict the behavior of other traffic actors surrounding it, plan trajectories and finally execute the motion [4].

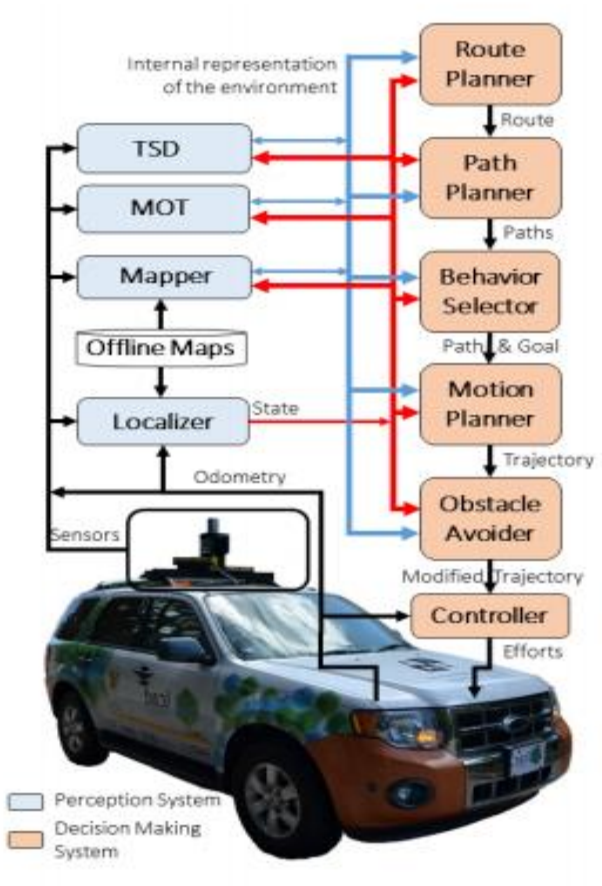

Figure 2: Self-Driving car Architecture [5]

1.3.1 Sensors

The sensor subsystem consists of several sensors that gather data about the surroundings of an autonomous vehicle. Some of the most common sensors employed for this purpose are as follows:

- Camera: Camera(image sensor) is certainly the most important sensor in an AV. Typically AVs have multiple cameras involved to provide a 360-degree view of the surrounding environment. Cameras have high resolution, are cheap, can collect a lot of data, therefore are useful for Deep Learning.

- Radar: Radars are again very common automotive sensors for object tracking and detection. Radars are cheap, do well in poor weather as well, but have low resolution.
- Lidar: Lidars are a bit expensive, but one of the most efficient sensors of an AV. Continuously rotating Lidar system sends thousands of laser pulses every second. These pulses collide with the surrounding objects and reflect, creating a 3D point cloud formation.

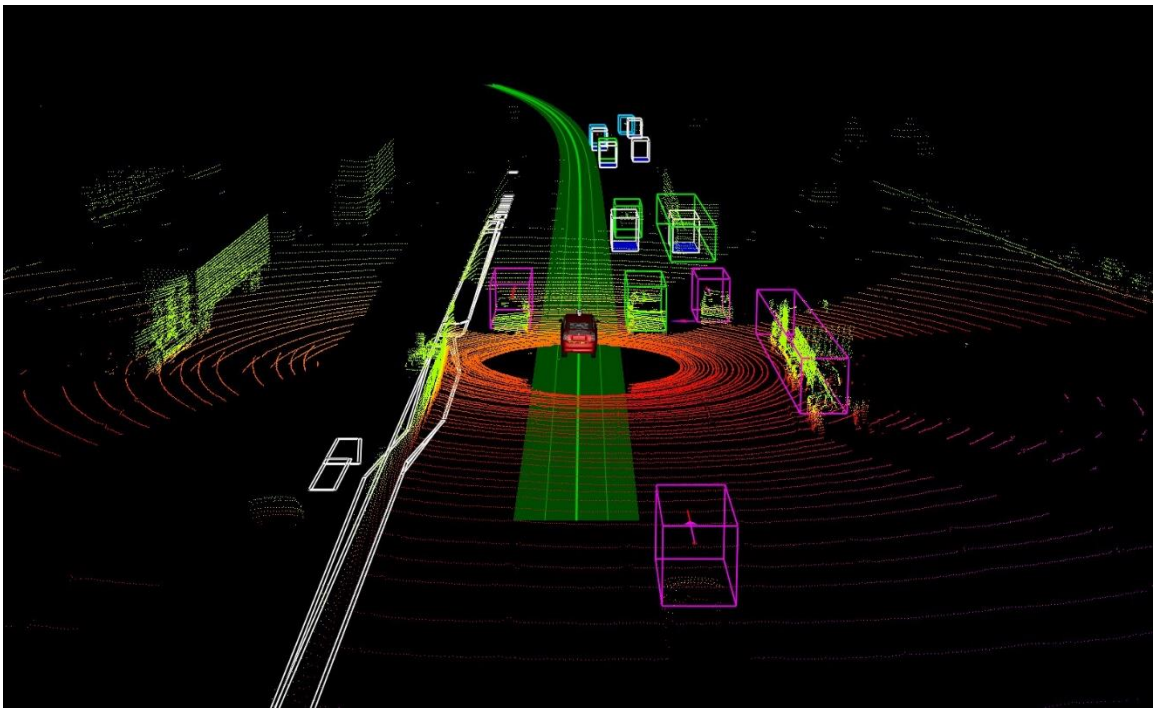

Figure 3: Object Detection using Lidar point clouds [6]

- GPS: GPS sensors are the common positioning sensors that give latitude and longitude information.
- Others: Apart from these, there are several other sensors like ultrasonic Sonars, IMUs, gyroscopes, etc. used in AVs [4].

1.3.2 Perception

Perception in AVs is responsible for estimating the state of the car and for creating an internal representation of the environment, using data captured by on-board sensors as well as the prior information about the sensors' models, road networks, traffic rules, car dynamics, etc. [5]. As described in [4], perception can be categorized into two components, as follows:

- Localization: As the name suggests, it involves localizing an AV within the driving scenario. This system uses the data from GPS and other sensors to estimate the AV's pose relative to the driving scenario. Usually, this is one of the initial steps of the autonomous driving process.
- Detection: This system gathers and processes the data from on-board sensors like camera, Lidar, radar, etc. to detect the static, variable, and dynamic objects in the driving scenario. However, the data from different on-board sensors need to be synchronized and processed accordingly to extract necessary information regarding the driving scenario. Sensor fusion is an approach for combining data delivered from disparate sources such that the coherent information is created. The resulting information is more certain than it would be possible when these sources were used individually. For example, on the AV, it is important to have a camera in order to clone a human vision, but the information about obstacle distance will be best gained through the sensors like radar or Lidar. For that reason, sensor fusion of camera with Lidar or radar data is very important since they are complementary [7].

1.3.3 Path Planning

The Path Planning functionality takes information from the perception system and uses it for long and short-range planning. There are several components of the Path Planning functionality such as the Route planner that plans the path that a vehicle should take between the given two points on a

map, the Prediction component that predicts behavior of other traffic actors, the Behavior planner that plans the behavior of the AV itself such as keeping the existing lane, change lanes, apply brakes or accelerate as needed, and finally the Trajectory planner that decides the ultimate trajectory that the AV must follow.

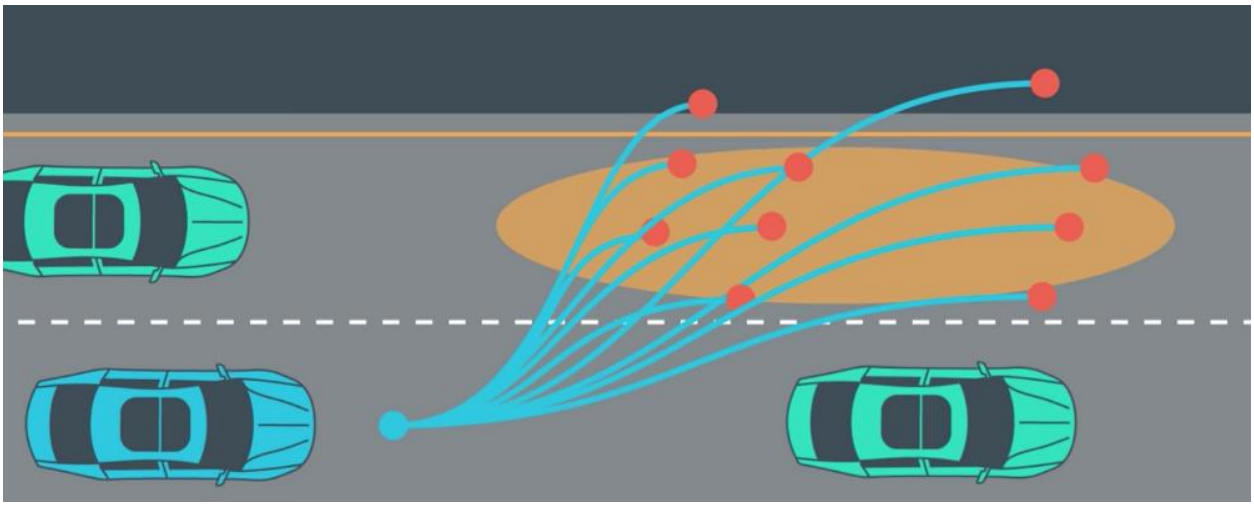

Figure 4: Path Planning in AVs [8]

1.3.4 Control

The Controller module receives the trajectory generated by the Path Planner and sends effort commands to the actuators of the steering wheel, throttle, and brakes of the AV to make the car execute the trajectory as best as the physical world allows [5]. Several controllers are used in AVs, depending on the problem to be solved. The PID (Proportional Integral Derivative) and MPC (Model Predictive Control) are two of the most commonly used controllers.

1.4 Testing and Validation of Autonomous Vehicles: A Challenge

As described in [9], recent breakthroughs in deep learning have accelerated the development of autonomous vehicles: many research prototypes now operate on real roads alongside human drivers. While advances in computer vision techniques have made human-level performance

possible on narrow perception tasks such as object recognition, several fatal accidents involving AVs underscore the importance of testing whether the perception and control pipeline – when considered a whole system, can safely interact with humans. Unfortunately, testing the AVs in real environments, the most straight forward validation framework for system-level input-output behavior requires prohibitive amounts of time due to the rare nature of serious accidents. Concretely, a recent study [10] argues that AVs need to drive "hundreds of millions of miles and, under some scenarios, hundreds of billions of miles to create enough data to clearly demonstrate their safety". As a result, AV developers test extensively on public roads, potentially putting other road users at risk. In one such unfortunate event, Elaine Herzberg, 49, was killed by an Uber test vehicle on 18th March 2018 in Tempe, Arizona [11]. Such incidents can infuse a sense of insecurity in the minds of common road users making the social acceptance of AVs a complex process. Aside from safety concerns, costs pose an additional challenge to the testing and validation of AVs. Each new configuration of the AV requires re-calibration of a physical vehicle, which is labor-intensive. Furthermore, the vehicle can only be tested under conditions limited by either a testing track or current traffic conditions if a public road test is being performed. Apart from that, the Machine Learning techniques used for Autonomous Vehicle algorithms rely on substantial amounts of annotated data in regular, as well as dangerous scenarios. The dataset must encompass varied weather and lighting conditions. Gathering such data by physical tests can be expensive, difficult, and even dangerous, as discussed above [12].

1.5 Simulation: An Effective Solution

The testing problem of the AVs can be transferred over to the virtual world, i.e., Simulation Environments. Realistic simulation environments comprising of high level-extensible modules like Environment module, Vehicle module, Physics engine, Sensor module, etc. provide an

efficient solution for testing and validation of Autonomous Vehicles. Modeling and simulation are well-established tools for analysis, design, acquisition, and training in the automotive domain. Despite the heterogeneity of subsystems and disciplines involved in the development of an Autonomous Vehicle, there are many simulation models that allow coverage of the entire development process [13]. Modern and dedicated simulators cover many aspects of the Autonomous Vehicle development process such as scenario generation, data gathering, realistic physics laws, realistic traffic flow, and photo-realistic graphics.

Figure 5: Waymo's simulation platform

The Automotive and tech giants in the self-driving industry have widely adapted simulation environments for testing and validation of the AVs. However, this doesn't completely eradicate

the need for on-road testing. Alphabet subsidiary Waymo has accumulated the most virtual mileage of all self-driving companies, with a February 2018 total of nine years and five million miles. Waymo simulations created over 2.5 billion self-driving miles in 2016 alone. The Waymo simulator (Carcraft) transforms real-world scenarios into virtual formats and runs 25,000 virtual cars simultaneously. The massive data flow from this process assists engineers in locating bugs and adjusting models efficiently [14]. Apart from that, Apollo (Baidu), XVIZ (Uber), AVS (GM Cruise), VIRTTEX (Ford), Nvidia Driveworks, etc. are some of the premium driving simulators used in the self-driving industry. Self-driving simulators can boost the speed of data collection to reach mileage accumulation targets while reducing fleet operating costs. Among all the rivals in the self-driving car industry, Waymo clearly appears to be leading the race. For this, a fair amount of credit goes Waymo's simulation platform as it has driven more virtual miles than any other competitor. Apart from that, CARLA, Microsoft AirSim, VisSim, CarSim, Gazebo, TORCS, Udacity simulator, Autono Vi-sim, etc. are some of the most widely used open-source simulators for Autonomous Driving research.

1.6 Reality Gap and Domain Randomization

However, the usage of simulation environments in Autonomous Vehicle research comes with its own complications. Models trained purely on synthetic data will fail to generalize to the real world, as there is a difference between simulated and real environments, in terms of both visual and physical properties. This difficulty of transferring simulated experience into the real-world is called the "Reality Gap". In order to efficiently use simulation environments for Autonomous Vehicle research, it is very important that this reality gap is bridged. Improving the photo-realism of simulation environments has been state-of-the-art when it comes to reducing the reality gap. However, for simulators with low-quality renderings, the object classes, their poses, number of

objects and other features are randomized while generating data. While training on such data, this randomness forces the neural network to go through a lot of relevant and irrelevant variety within the data. Hence, such a neural network model learns to identify relevant information in the frame at the same time, knowing what to ignore. When such a model is tested on real-world data, the real world may appear as just another variation. This technique is called Domain Randomization, which has emerged as a worthy option to bridge this reality gap and until now, has been mostly applied to basic Object Recognition tasks. The details of Domain Randomization are further discussed in section 2.5.1.

Chapter 2: A Literature Review

This chapter starts with a brief discussion about the Computer Vision techniques used for Autonomous Vehicle research, and the traditional datasets used to facilitate it. After that, we discuss the drawbacks and flaws associated with those traditional datasets. Then, we discuss how synthetic data has emerged as an option to fill this data void and the simulation environments used for that purpose in detail. Furthermore, we discuss how the synthetic data generated from basic simulators face the problem of the Reality gap and the ways to reduce this Reality gap.

2.1 Computer Vision in Autonomous Vehicles

Various Machine Learning and Deep Learning based techniques are used for several tasks like Object Detection, Object Recognition, Motion prediction, and Risk Assessment. Some of the most efficient techniques used for Autonomous Driving are briefly described as follows:

2.1.1 Faster R-CNN

Faster R-CNN [15] is one of the most widely used techniques for Object Detection in Autonomous Driving research. Faster R-CNN has displayed a superior performance over its predecessors, R-CNN [16] and Fast R-CNN [17]. At the conceptual level, Faster R-CNN is composed of 3 neural networks – Feature Network, Region Proposal Network (RPN), and Detection Network [18].

- The Feature Network is usually a well known pre-trained image classification network such as VGG [19] minus a few top/last layers. The function of this network is to generate good features from images. The output of this network maintains the shape and structure of the original image (i.e., still rectangular, pixel size, etc.).
- The Region Proposal Network (RPN), is usually a simple network with three convolutional layers. There is one common layer that feeds into two layers – one for classification and

the other for bounding box regression. The purpose of RPN is to generate a number of bounding boxes called Region of Interests (ROI) that have a high probability of containing any object. The output from this network is a number of bounding boxes identified by the pixel co-ordinates of two digital corners, and a value (1, 0 or -1) indicating whether an object is in the bounding box, or not in the bounding box or the box can be ignored respectively.

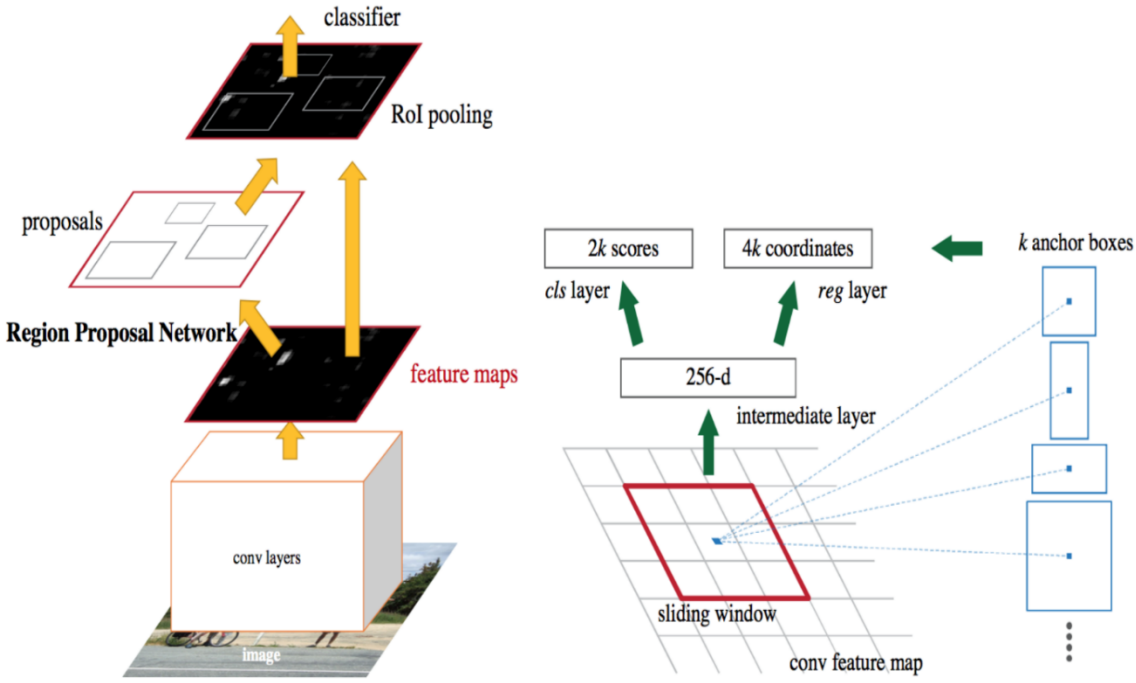

Figure 6: Faster R-CNN [15]

- The Detection Network (sometimes also called the RCNN network) takes input from both of the above-discussed networks and generates the final class and bounding box. It is normally composed of 4 Fully Connected or Dense layers. There are two stacked common layers shared by a classification layer and a bounding box regression layer. To help it

classify only the inside of bounding boxes, the features are cropped according to the bounding boxes.

2.1.2 You Only Look Once (YOLO)

YOLO [20] is an algorithm that utilizes a single convolutional network for object detection. Unlike other object detection algorithms that sweep the image bit by bit, the algorithm takes the whole image and reframes the object detection as a single regression problem, straight from pixels to bounding box co-ordinates and class probabilities. YOLO trains on full images and directly

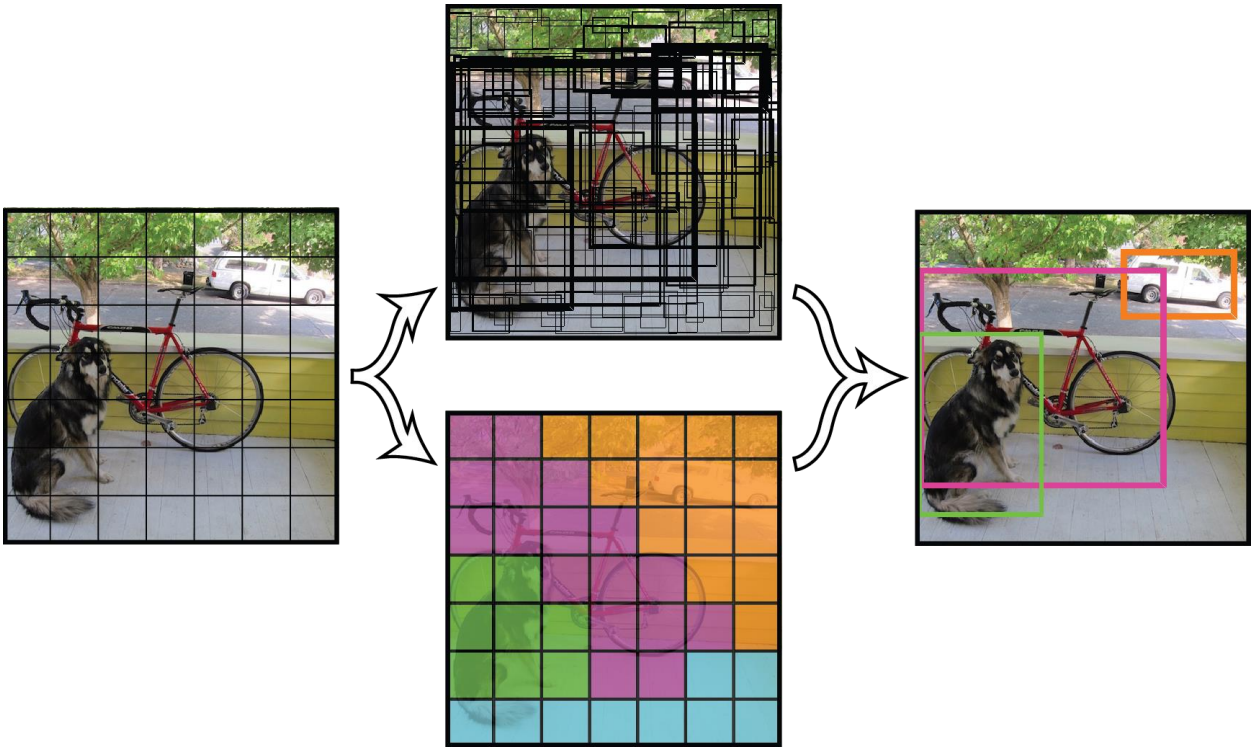

Figure 7: YOLO: Object Detection with YOLO [20]

optimizes detection performance. YOLO divides up the image into a grid of 13 by 13 cells. Each of these cells is responsible for predicting five bounding boxes. YOLO outputs the confidence

score that tells us how certain it is that the predicted bounding box actually encloses some object. After that, for each bounding box, the cell also predicts an object class.

2.2 Data Limitations for ComputerVision

Datasets are an integral part of contemporary object recognition research. They have been the chief reason for the considerable progress in this field, not just as the source of large amounts of training data, but also as a means of measuring and comparing the performance of competing algorithms [21]. However, training deep neural networks for computer vision tasks typically require large amounts of labeled training data. A variety of approaches have been proposed to efficiently label large amounts of training data, including crowdsourcing, gamification, semi-supervised labeling, and Mechanical Turk. These approaches remain fundamentally bounded by the amount of human effort required for labeling or supervision.

2.2.1 Traditional Datasets Used in Computer Vision

Traditionally, datasets like KITTI [22], CityScapes [23], Imagenet [24], MS COCO [25], CIFAR-10, Open Images, etc. have been the most popular open-source datasets used in the computer vision field.

- The KITTI dataset: The KITTI dataset has been recorded from a moving platform while driving around Karlsruhe, Germany. It includes camera images, laser scans, high-precision GPS measurements, and IMU accelerations from a combined GPS/IMU system. The data in this dataset is calibrated, synchronized, and timestamped, along with rectified and raw image sequences. The dataset also contains object labels along with online benchmarks for stereo, optical flow, object detection, and other tasks. For each dynamic object within the reference camera's field of view, the annotations are provided annotations in the form of

3D bounding boxes, represented in Velodyne coordinates. The object classes covered are ‘car,’ ‘van,’ ‘Truck,’ ‘Pedestrian,’ ‘Person(sitting),’ Cyclist,’ Tram’ and ‘Misc’(e.g., Trailers, Segways, etc.).

Figure 8: Examples from KITTI dataset [22]

- The CityScapes dataset: CityScapes is comprised of a large, diverse set of stereo video sequences recorded in streets from 50 different cities. 5000 of these images have high-quality pixel-level annotations; 20,000 additional images have coarse annotations to enable methods that leverage large volumes of weakly labeled data. However, recording in adverse weather conditions such as heavy rain or snow was avoided deliberately. The

densely annotated data of 5000 images is split into separate training, validation, and test sets.

Figure 9: An example from CityScapes Dataset [23]

- The ImageNet database: ImageNet database is a large visual database designed for use in visual object recognition software research. It contains more than 14 million images that have been hand-annotated to indicate what objects are pictured in at least one million pictures, and bounding boxes are also provided. ImageNet contains more than 20,000 categories with any typical category, such as “balloon” or “strawberry,” consisting of several hundred pictures. The database of annotations of third party URLs is freely available directly from ImageNet, though the actual images are not owned by ImageNet. Since 2010, the ImageNet project runs an annual software contest, the ImageNet Large

Scale Visual Recognition Challenge (ILVRC), where software programs compete to classify and detect objects and scenes. ImageNet crowdsources its annotation process. Image level annotations indicate the presence or absence of an object class in an image such as “there are tigers in this image” or “there are no tigers in this image.” Object-level annotations provide a bounding box around the indicated object.

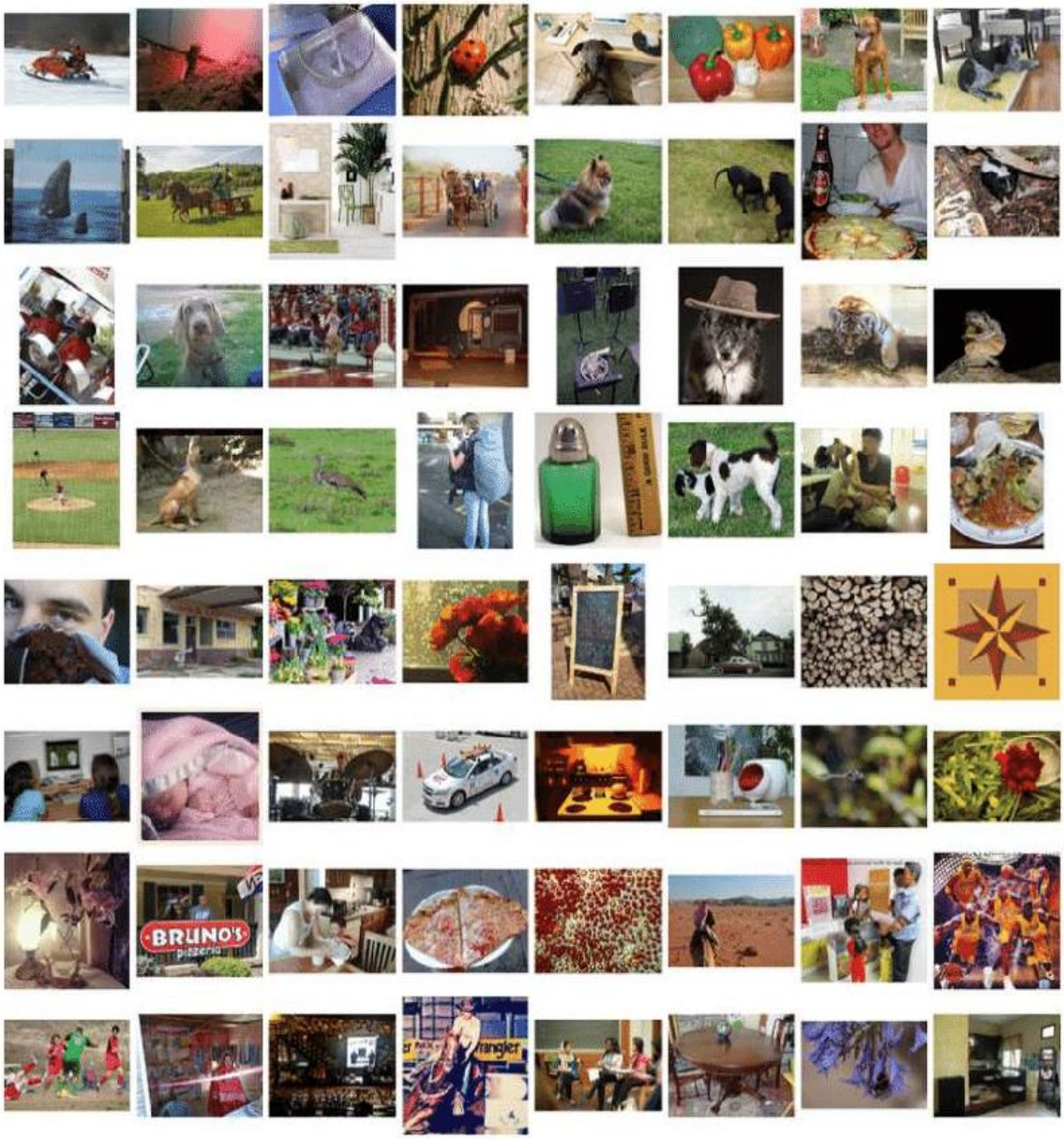

Figure 10: Examples from ImageNet dataset [24]

- Microsoft COCO: MS COCO is another such large-scale dataset with the goal of advancing state-of-the-art in object recognition by placing the question of object recognition in the context of the broader question of scene understanding. This dataset contains photos of 91 object types and has a total of 2.5 million labeled instances in 328k images. Such annotation drew upon extensive crowd worker involvement via novel user interfaces for category detection, instance spotting, and instance segmentation.

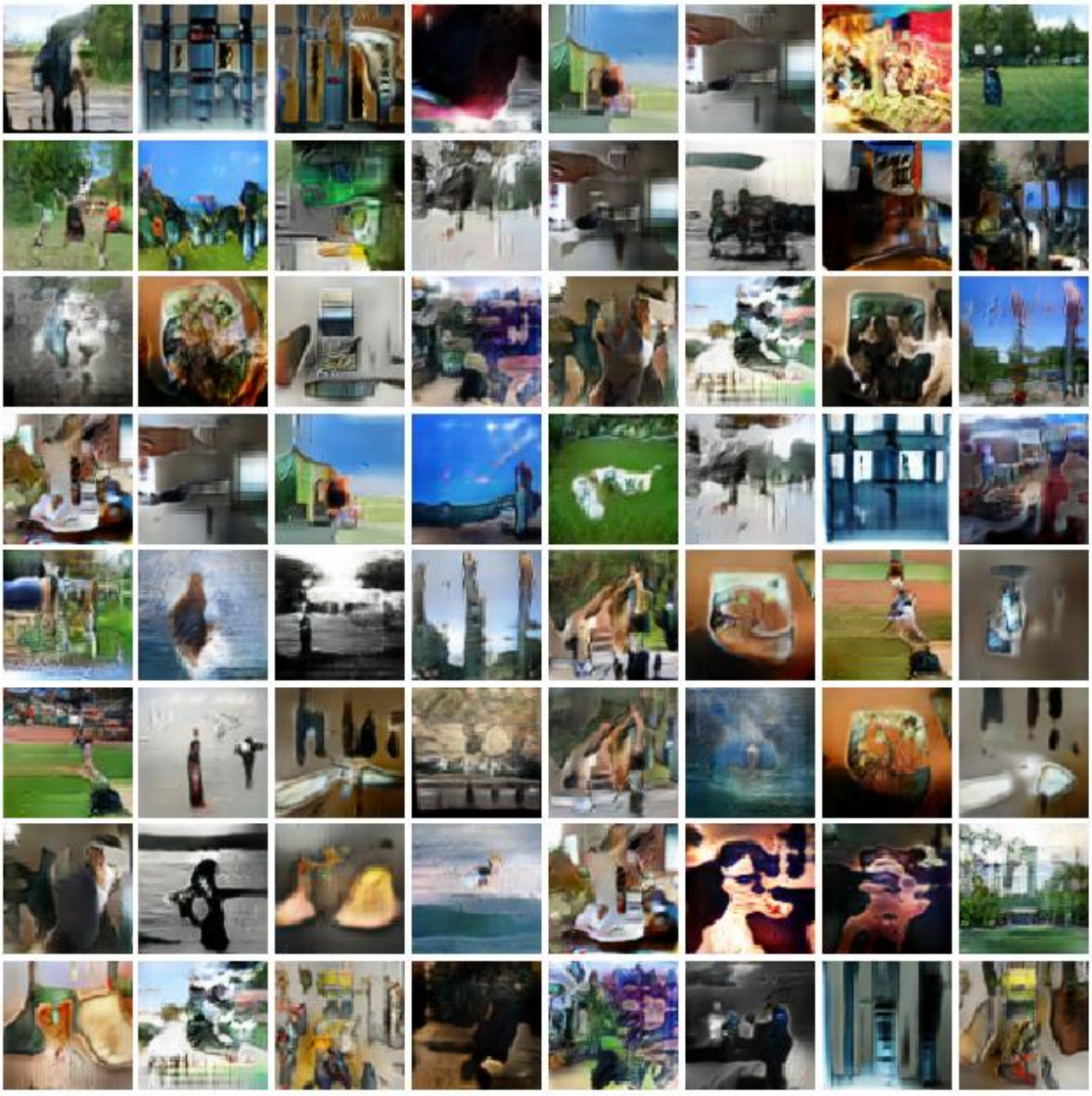

Figure 11: Examples from MS COCO dataset [25]

2.2.2 Need for Datasets Improvement

It would be safe to say that we are in the midst of a data revolution. Ubiquitous access to image datasets has been responsible for much of the recent progress in object recognition after decades of proverbial wandering in the desert. For instance, it was the availability of face training data, more than perceived advances in machine learning that produced the first breakthrough in face detection [26]. And it is the dataset of millions of photographs of consumer products, as much as clever feature matching that allowed visual search engines like GOOGLE GOOGLES to become a reality. However, like any proper revolution, this one too has brought with it new problems to replace the old ones. It appears that this field is now getting too obsessed with evaluation, spending more time staring at precision-recall curves than at pixels [21]. Some of the evident issues with current state-of-the-art datasets are as follows:

2.2.2.1 Lack of Diversity

Capturing sufficient diversity in a dataset is a challenge. We can often observe that datasets are restricted to a selected subset of cases, each dataset tackling one small part of the whole set of possible environments and conditions: for example, KITTY [22] and CityScapes [23] are collected only in Germany or even the Oxford [27] dataset is only collected in Oxford, meaning that those datasets are geographically restricted.

2.2.2.2 Dataset Bias

The visual world is so complex and nuanced that any finite set of samples ends up describing just some of its aspects. Moreover, in case the samples are collected for a particular task, they will inevitably cover just some specific visual region. Hence, it is not surprising that pre-defined image

collections like existing computer vision datasets, present such specific bias to be easily recognizable. The main types of bias [21] found in existing computer vision datasets are as follows:

- Selection Bias: It is a known fact that datasets that are gathered automatically fare better than those collected manually. However, getting images from the internet does not guarantee a fair sampling, since keyword-based searches will return only particular types of images. Obtaining data from multiple sources (e.g. multiple search engines from multiple countries) can somewhat decrease selection bias.
- Negative Bias: Having a rich and unbiased negative set is important to classifier performance. Therefore, datasets that only collect the things they are interested in might be a disadvantage, because they are not modeling the rest of the visual world. An effective remedy would be to add negatives from other datasets.
- Capture Bias: Professional photographs, as well as photos collected using keyword search, appear to suffer considerably from the capture bias. The most well-known bias is that the object is almost always in the center of the image. For example, searching for a "mug" on Google Image search will mostly provide images with mugs situated at the center of the frame. Also, it will reveal another kind of capture bias: almost all the mugs have a right-facing handle.

2.2.2.3 Lack of challenging Weather and Lighting conditions

In computer vision tasks for applications like Autonomous Driving, it is very crucial that Computer vision algorithms are also trained in challenging and variety of weather and lighting conditions. Apart from a few datasets, all others only have images of driving scenarios in regular weather conditions. However, the landscape, roads, visibility, wind, etc. will differ for different parts of

the world and with varying seasons as well. It is so important that Autonomous Vehicles are trained for every such scenario before finally deploying them on roads for public usage.

2.2.3 Synthetic Data: An Option

Amidst such complications with using real-world data, researchers have looked for efficient alternatives and Synthetic data (created using Virtual Reality and/or Augmented reality) has emerged as a worthy option. A promising approach to generate synthetic data is to use a graphic simulator to generate automatically annotated data. Several such simulated datasets have been created in recent years as found in [28], [29], [30], [31], etc.

- SYNTHIA dataset: SYNTHIA stands for the SYNTHetic collection of Imagery and Annotations. This dataset is one of the best examples of datasets generated using simulated environments for Autonomous Driving research. SYNTHIA consists of photo-realistic frames rendered from a virtual city and comes with precise pixel-level semantic annotations for thirteen classes, i.e., sky, building, road, sidewalk, fence, vegetation, lane-marking, pole, car, traffic signs, pedestrians, cyclists and miscellaneous.

Figure 12: Examples from SYNTHIA Dataset [31]

2.3 Simulation Environment

For dataset generation regarding Autonomous Driving research, there are various dedicated as well as general-purpose simulators available. Some of the dedicated simulators (Driving Simulator) even allow training the Autonomous Vehicles in the simulated environment, apart from data generation. Driving Simulators are usually a collection of high-level extensible modules that allow the rapid development and testing of vehicle configurations and facilitate the construction of complex traffic scenarios. They support multiple vehicles with unique steering or acceleration limits, as well as unique tire parameters and dynamic profiles to name from the vast features they provide. Engineers can specify the specific vehicle sensor systems and vary the time of day and weather conditions to generate robust data. Non-vehicle participants such as cyclists and pedestrians can be assigned specified routes or script scenarios that place the ego vehicle in dangerous reactive scenarios [12]. Some of the main benefits of using driving scenarios are discussed as follows:

- Data Generation: Driving simulators can generate virtually unlimited data for research and testing purposes. Apart from the main scene images, they also allow to export different vehicle configuration data and the data from virtual sensors attached to the ego vehicle. Furthermore, the data generated is automatically annotated as programmed, which eliminates the need for the labor-intensive manual annotating process.
- Varying vehicle, cyclist, pedestrian, and traffic conditions: The driving simulators include various vehicle and sensor models, pedestrians, and cyclists, as discussed above. The diversity of these traffic actors allows training for classification on different shapes, sizes, colors, and behaviors of cyclists, pedestrians, and other drivers.

- Dynamic Traffic, Weather, and Lighting Conditions: The driving simulators provide high fidelity traffic simulation, supporting dynamic changes in traffic density, time of day, lighting, and weather, including rain and fog.
- Rapid Scenario Construction: Typical road networks can be easily laid out using the in-built tools and are automatically connected for routing and navigation purposes.

2.3.1 Components of a Driving Simulator

An ideal simulator for autonomous driving research comprises a variety of inter-linked components covering the navigation, perception, and control modules. They are discussed as follows:

- Game Engine (Rendering Engine): A game engine is a part of a computer game that contains a 2D or 3D graphic representations (rendering engine), representations of physical laws (Physical Engine), or collision detection (and collision response), sound, scripting, animation, artificial intelligence, networking, streaming, memory management, threading, localization support, scene graph and may include video support for cinematics. The most modern game engines also include support for Virtual Reality (VR) simulation. However, for Driving Simulators, only the Rendering Engine is used among all the above-mentioned features of a game engine. This is due to the reason that the physics engine or collision response of a game engine may not be up to the standard required for a high-fidelity simulator. Some of the most popular game engines used for building driving simulators are Unreal Engine 4 [32], Unity 3D [33], Blender [34], CryEngine [35], etc.

  As of now, Unreal Engine 4, provided by Epic Games, has emerged as a favorite game engine for our purpose. At the same time, Unity 3D is also improving at an impressive rate.

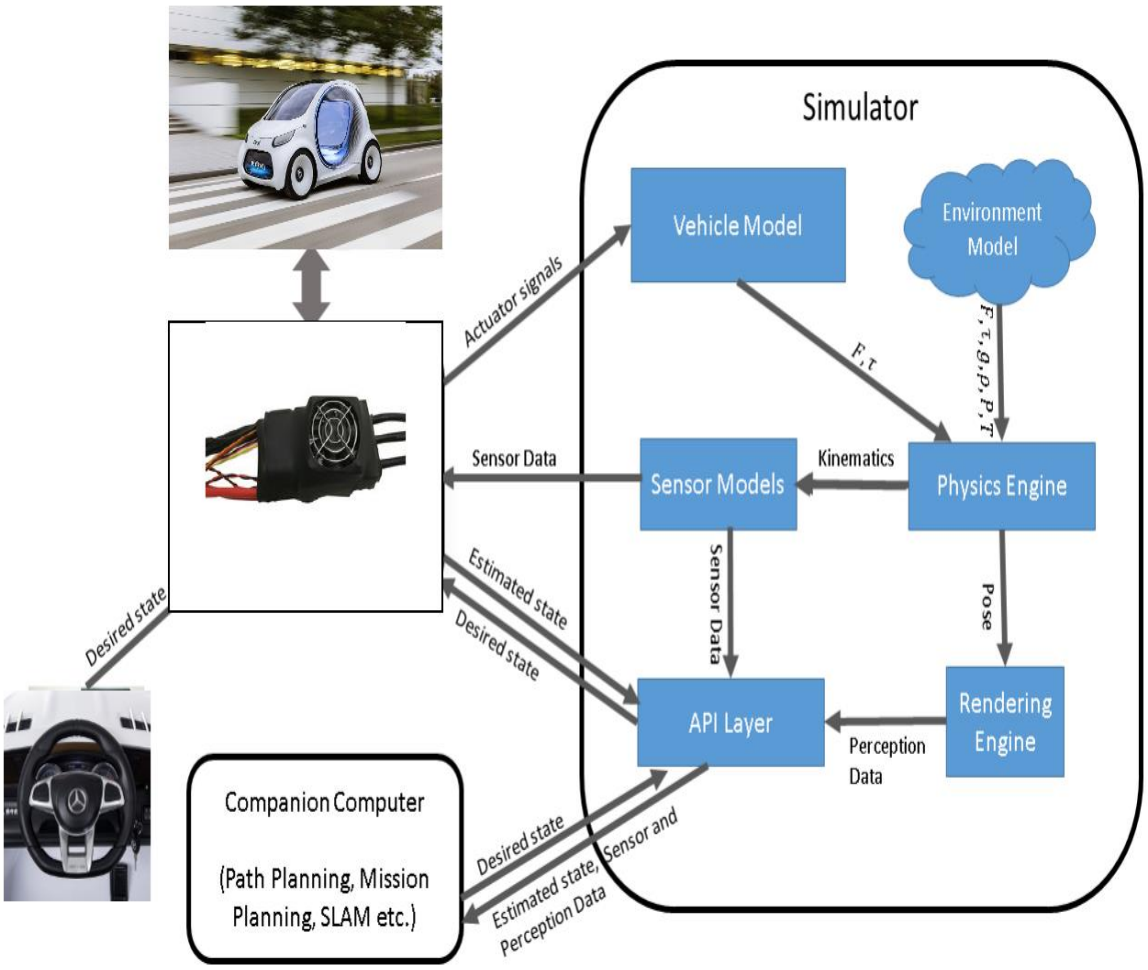

Figure 13: Components of a Driving Simulator [36]

- Physics Engine: As the name suggests, a Physics Engine is a computer software that provides an approximate simulation of certain physical systems such as rigid body dynamics (including collision detection and response), soft body dynamics, and fluid dynamics. A Physics Processing Unit (PPU) is a dedicated microprocessor to host the physics engine.

- Environment Model: This module allows a user to specify a variety of lighting and weather conditions. The importance of data with varying weather and lighting conditions has already been discussed earlier.
- Vehicle Model: This module contains a variety of vehicle templates that can be simultaneously spawned in the simulation environment. Realistic vehicle control and behavior such as key steering assistance, braking assistance, support of traction control, flexible tires simulation, tire types, and customization, etc. are encoded in the vehicle templates.
- Sensor Model: This one of the most important and unique modules of a driving simulator. The sensor model comprises various virtual sensors that can be simultaneously equipped on the vehicles in the simulation environment. Though virtual, these sensors must be high fidelity and also behave like any other realistic sensors in the real world. The sensors are expected to provide realistic inputs to the ego vehicles, that can be extracted and stored with automatic annotations for research purposes. Some of the common sensors provided in driving simulators are as follows:
    - Camera
    - Radar
    - Lidar
    - GPS
    - Ultrasonic sensor
- API Layer: An API Layer is the component that makes the driving simulator extensible. Using simple but powerful scripting languages, a user can control traffic actors, weather and lighting conditions, change roadmaps, extract and process data, etc. Usually, this is

implemented as a client-server architecture where the game engine based simulator acts as a server, and various python/C++ scripts are on the client-side.

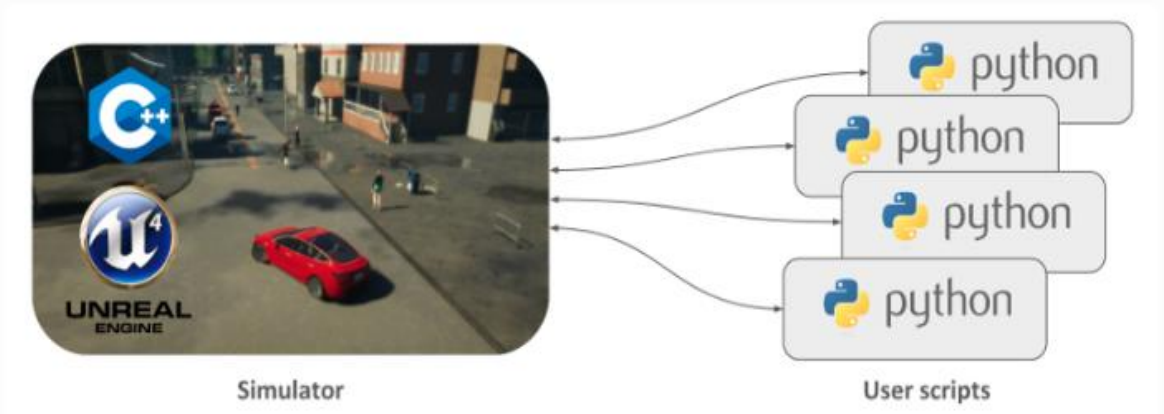

Figure 14: Client-Server architecture in Driving Simulators [37]

2.4 Various Driving Simulators

Some of the lastest and efficient driving simulators and their working is discussed as follows:

2.4.1 CARLA: An Open Urban Driving Simulator

CARLA [37] has been developed from the ground up to support the development, training, and validation of autonomous driving systems. In addition to open-source code, and protocols, CARLA provides open digital assets (urban layouts, buildings, vehicles) that were created for this purpose, and can be used freely.

CARLA consists mainly of two modules, the CARLA Simulator and the CARLA Python API module. The simulator does most of the heavy work, controls the logic, physics, and rendering of all the actors and sensors in the scene; it requires a machine with a dedicated GPU to run. The

Figure 15: A street in Town 2 in four weather conditions. [37]

CARLA Python API is a module that you can import into your python scripts, and it provides an interface for controlling the simulator and retrieving data. Most aspects of the simulation are accessible through the Python APIs, and the remaining will be covered in future releases. CARLA has been built on the Unreal Engine 4 (UE4). Apart from that, CARLA comprises almost all modern features and functionalities (mentioned in section 2.4.1) like impressive vehicle models, environmental models, photo-realistic 3D objects (static, variable, and dynamic), virtual sensors, etc. For implementation purposes, the necessary information and basic steps to get started with this simulator can be found in [38]. Recently, various works have been implemented using the CARLA simulator for different purposes like vehicle testing, data generation, sensor validation,

object detection, semantic segmentation, etc. and can be found in [39], [40], [41], [42], [43], [44], etc.

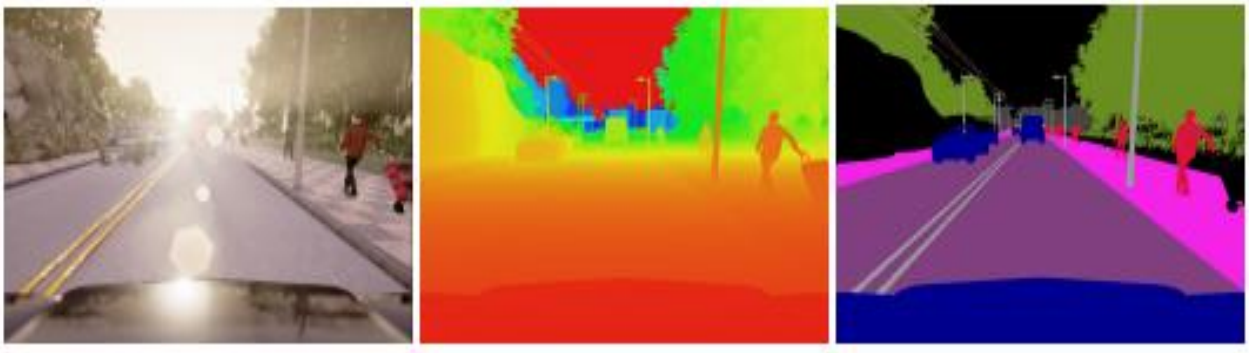

Figure 16: Three of the sensing modalities provided by CARLA. From left to right: normal vision camera, ground-truth depth, and ground-truth semantic segmentation. [37]

2.4.2 Microsoft AirSim

AirSim [36] is another high-fidelity simulation platform developed with a goal of encouraging AI research to experiment with deep learning, computer vision, and reinforcement learning algorithms for autonomous vehicles. It is a simulator for drones, cars, and more built on Unreal Engine and includes a physics engine that can operate at a high frequency for real-time hardware-in-the-loop (HITL) simulations with support from popular protocols (e.g., MavLink). This simulator is designed from the ground up to be extensible to accommodate new types of vehicles, hardware platforms, and modular protocols. It is developed as an Unreal plugin that can be simply dropped into any Unreal environment. As a whole, AirSim comprises state-of-the-art modules like the modern physics engine, environment model, vehicle model, sensor model and rendering engine (Unreal Engine). For implementation purposes, the necessary information and basic steps to get started with this simulator can be found in [45].

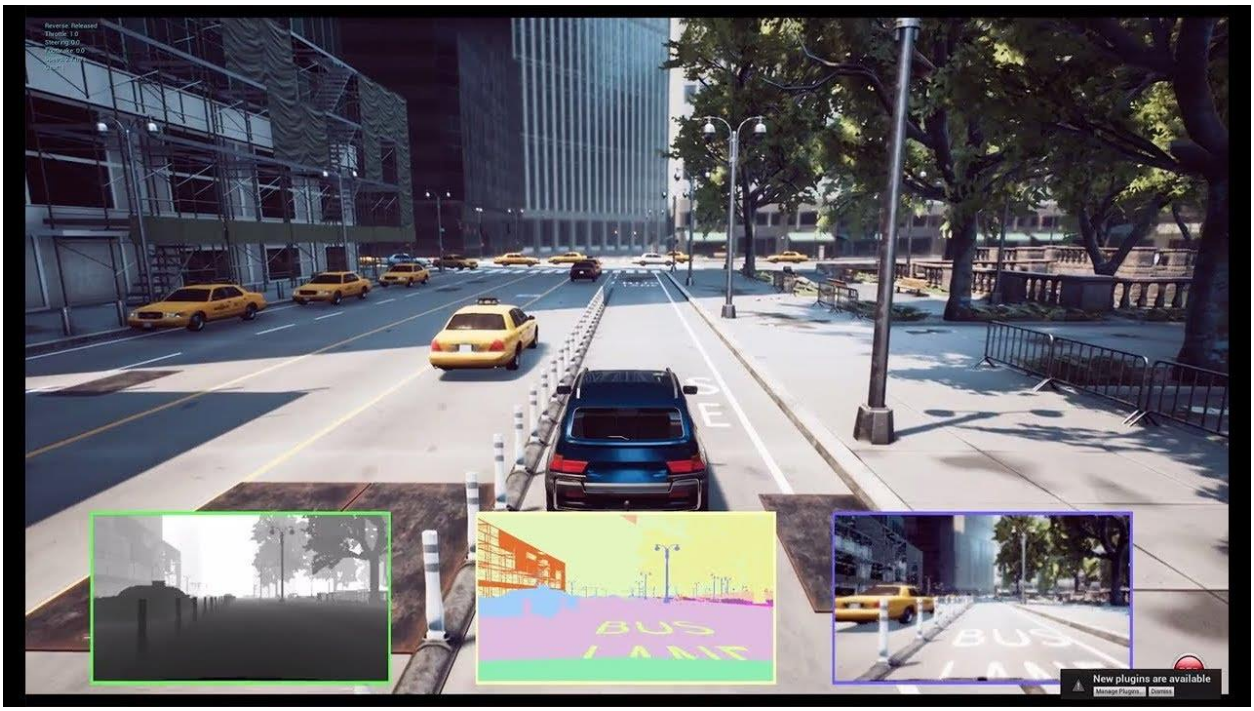

Figure 17: Sample road scene in AirSim [46]

2.4.3 Autono Vi-Sim

Autono Vi-Sim [12] is another such high-fidelity simulation platform for autonomous driving data generation and driving strategy testing. It is also a modern state-of-the-art driving simulator designed to allow researchers and engineers to rapidly configure novel road networks, and to test these in a variety of weather and lighting conditions.

Best et. al in [12], developed Autono Vi-Sim and created various complex testing driving scenarios as follows:

- Passing a bicycle

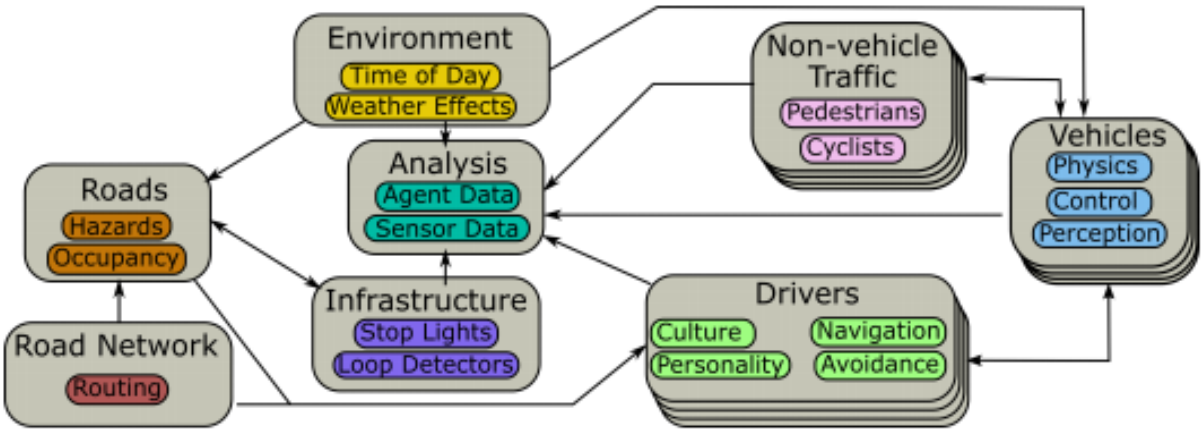

Figure 18: Autono Vi-Sim Architecture [12]

- Jaywalking pedestrian
- Sudden stop at high speed
- High-density traffic approaching at a turn
- Car suddenly entering Roadway
- S-turns

As shown in the figure below are some sample scenes from Autono Vi-Sim: (A): Heavy fog obstructs the view of a vehicle. (B): the entire simulated city. (C): Vehicles pass through a slick intersection in rainy conditions

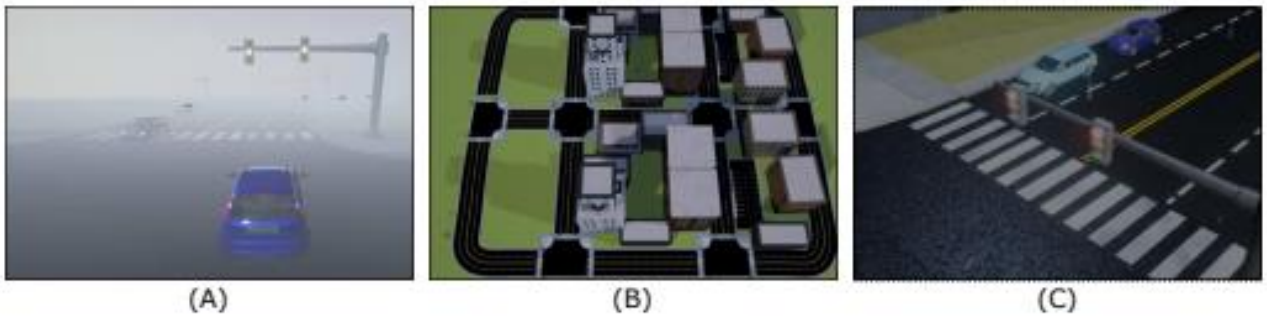

Figure 19: Samples from Autono Vi-Sim [12]

2.4.4 Comparison of existing Driving Simulators

Simulator	License	Physics Engine	Graphic Engine	Scripting Language	Geographically Diverse	Curbs Dataset Bias	Challenging Weather conditions
CARLA [37]	GPL/Open Source	Unreal Engine	GPU	Python	No	Yes	Yes
AirSim [36]	GPL/Open Source	Unreal Engine	u	C++, Python, C#, Java	No	Yes	Yes
DeepDrive [47]	GPL/Open Source	Unreal Engine	u	C++, Python	No	No	Yes
Udacity [48]	GPL/Open Source	Unity	u	C++, Python	No	No	Yes
NVIDIA DRIVE Constellation [49]	Restricted	PhysX/CUDA	GPU	C/C++, Python	Yes	Yes	Yes
Carcraft (Waymo) [50]	Restricted	u	u	u	u	u	u
SIMLidar [51]	GPL/Open Source	u	u	C++	No	No	No
Helios [52]	GPL/Open Source	JMonkey Engine	OpenGL	Java	Yes	No	Yes
Autono Vi-SIM [12]	u	Unreal Engine	OpenGL	C++/Python	No	No	Yes
RADSim [53]	Commercial	u	u	MATLAB	No	No	No
SIMSonic [54]	GPL/Open Source	u	u	R	No	No	No

Table 1: Summary of the features of specific simulators for AVs. [13]

Table Legend: u-Unknown or could not be determined.

2.5 Reality Gap and Domain Randomization

So far, we have discussed how synthetic data generated from driving simulators can be a safe, inexpensive, and efficient alternative to real-world data collection. However, the challenge with simulated training is that even the best available simulators do not perfectly capture the reality. Models trained purely on synthetic data fail to generalize to the real world, as there is a discrepancy between simulated and real environments, in terms of both visual and physical properties [55]. This difficulty of transferring simulated experience into the real-world is called the "**Reality Gap**".

The reality gap is a subtle but important inconsistency between reality and simulation that prevents simulated robotic experience from directly enabling effective real-world performance. Bousmalis et. al. [55] clearly specify that visual perception often constitutes the widest part of the reality gap: while simulated images continue to improve in terms of fidelity, the peculiar and pathological regularities of synthetic pictures, and the wide, unpredictable diversity of real-world images, makes bridging the reality gap particularly difficult when the robot must use vision to perceive the world for most of the tasks.

In fact, the more we increase the fidelity of our simulations, the more effort we have to expend in order to build them, both in terms of implementing complex physical phenomena and in terms of creating other content (e.g., objects, backgrounds) to populate these simulations. This difficulty is constituted by the fact that powerful optimization methods based on deep learning are exceptionally proficient at exploiting simulator flaws: the more powerful the machine learning algorithm, the more likely is to discover how to "cheat" the simulator to succeed in ways that are infeasible in the real world.

Various works like [55], [56], [57], [58], [59], [60] and [61] have discussed the problem of Reality Gap in brief and provided solutions to bridge the Reality Gap. Almost all these approaches suggested the following ways to reduce the Reality Gap:

- Increase the resemblance between real and simulated domains by using high fidelity simulators to generate data (which we did in this work).
- Customize the generated simulated data to make them look real using GANs.
- For low-quality synthetic images, use domain randomization to create enough variety in the data and then train the Neural Network.

Traditionally, increasing the fidelity of simulators and generating realistic data has always been state-of-the-art in the Computer Vision field. On the other hand, there has been a rise in the approaches where colossal variety is randomly infused into photographically low-quality data, such as Domain Randomization, Domain Adaptation, Structured Domain Randomization, etc. Such approaches are discussed in the following sub-section.

2.5.1 Domain Randomization

Bridging the "Reality Gap" that separates the simulated world from the real-world could accelerate robotics (including autonomous driving) research through improved data availability. Tobin et al. [56], introduced the concept of Domain Randomization, a simple technique for training models on simulated images that transfer to real images by randomizing rendering in the simulator. The key concept behind this is generating a vast amount of training data by randomizing the object classes, their poses, the number of objects in a frame, colors, textures, lighting conditions, etc. In this manner, the model trained on such data will be exposed to a wide range of environments and

scenarios. When such a model is tested on a real-world scenario, it will generalize to the real-world with no additional training, i.e. the real-world may appear to the model as just another variation.

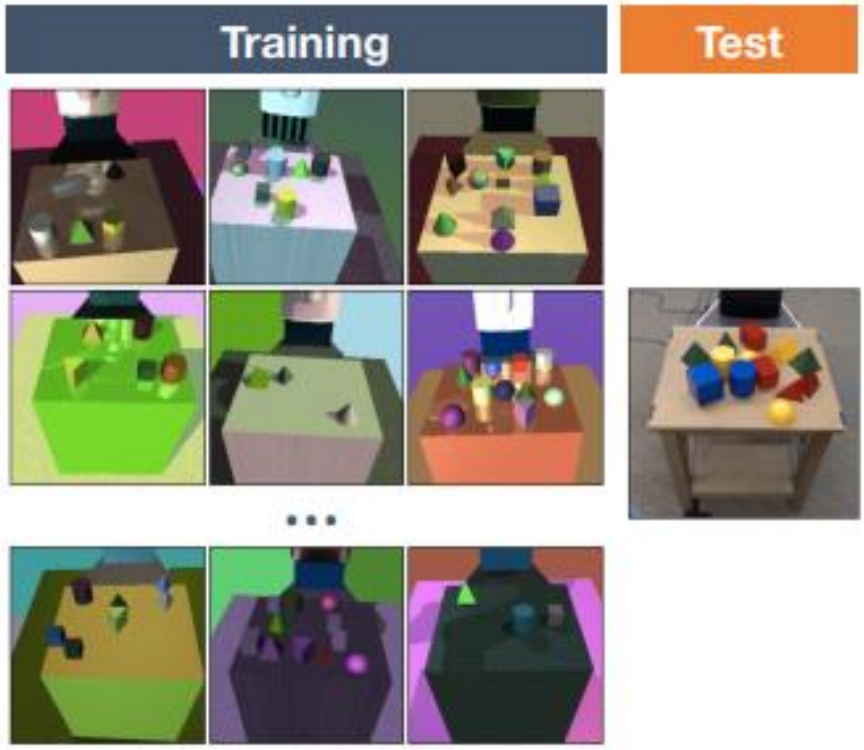

Figure 20: Training data generated using Domain Randomization [56]

In their approach, Tobin et al. [56], randomized the following aspects of each domain for each sample used during training (the models used to validate this approach were Object Detection models).

- Number and shape of distractor objects
- Position and texture of all objects
- Position, orientation, and field of view of camera
- Number of lights in the scene
- Type and amount of random noise in the images

Domain Randomization for Computer Vision tasks is still in its infancy and has been mostly used for basic shapes and objects like cubes, cylinders, pyramids, etc. as observed in [56] and [57]. However, Tremblay et al. [58] demonstrated that Domain Randomization is an effective technique to bridge the reality gap in the Autonomous Driving domain as well. Using synthetic DR data alone, they trained a neural network to accomplish tasks like object detection with performance comparable to more labor-intensive datasets. Their work proves that using DR to generate datasets for training deep neural networks is a promising approach to leverage the power of synthetic data.

Figure 21: Images from Virtual KITTI (first row) and DR approach (second row) [58]

Prior to Domain Randomization, researchers had resorted to Domain Adaptation, which aims to tailor the model for a particular target domain by jointly learning from the source synthetic data and the data of the target real domain. This would not particularly work for Autonomous Vehicles

as it is almost impossible for a car manufacturer to know in advance under what domain (which city, what weather, day or night) the vehicle will be used. Due to such reasons, Domain randomization seems to be an ideal option to bridge the reality gap in Autonomous Vehicle research [59].

2.6 Related Works

The table below is a compilation of the research works most relevant to the domains covered in this thesis.

Title	Accomplishments	Limitations
Unbiased Look at Dataset Bias, by A. Torralba, A. Efros, 2011	--This paper conducts a survey of the current state of object recognition datasets. --They present a comparison study of relative data bias, cross dataset generalization, etc. and other faults with existing datasets.	--This paper points out the shortcomings in current datasets used for Computer Vision but doesn't provide any effective solutions to improve them. --Apart from that, this is a fairly old survey and hence doesn't reflect the recent changes that have occurred in those datasets.
The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes, by G. Ros, L. Sellart, J. Materzynska, D.	--Presents SYNTHIA, a synthetic dataset of urban scenes, consisting of photo-realistic frames rendered from a virtual city with precise pixel-level annotations for popular classes like sky, building, road, sidewalk, pedestrians, vehicles, etc.	--They have portrayed and evaluated the SYNTHIA dataset specifically for Semantic Segmentation. --However, a dataset for Computer Vision needs to be general-purpose and mature enough for various tasks like

Vazquez, A. Lopez, 2016		object detection, object recognition, etc.
A Systematic Review of Perception System and Simulators for Autonomous Vehicles Research, by F. Rosique, P. Navarro, C. Fernández, A. Padilla, 2019	--This paper presents a systematic review of the perception systems and simulators for Autonomous Vehicles (AVs). --Conducts a survey comparing various open and non-open source driving simulators available for AV research.	--It doesn't cover the suitability of the surveyed simulators for specific tasks related to Autonomous Driving.
CARLA: An Open Urban Driving Simulator, by A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, V. Koltun, 2017	--Introduces CARLA, an open-source simulator for autonomous driving research. --It further explains the components of the simulator as well as the modern features that can be used to develop and train Autonomous Vehicle systems and then evaluate them in controlled scenarios.	--The initial stable versions of the simulator provided in this paper doesn't allow the usage of custom-built environments.
AutonoVi-Sim: Autonomous Vehicle	--Presents Autono Vi-Sim, a high-fidelity simulation platform for	--The simulator provided in this work is still in active

Simulation Platform with Weather, Sensing, and Traffic control, A. Best, S. Narang, L. Pasqualin, D. Barber, D. Manocha, 2017	autonomous driving data generation and driving strategy testing. --Furthermore, they use the simulator to generate various complex driving scenarios.	development and needs improvement to its physics engine as well as sensor models.
Training Deep Networks with Synthetic Data: Bridging the Reality Gap by Domain Randomization, by J. Tremblay, A. Prakash, D. Acuna, M. Brophy, V. Jampani, C. Anil, T. To, E. Cameracci, S. Boochoon, S. Birchfield, 2018	--This paper aims at bridging the reality gap in Computer Vision through the Domain Randomization technique. --First of all, they generate synthetic data applying Domain Randomization and use this data to train Object Detection models. --They further demonstrate that such a model works efficiently when tested on real-world data.	--This approach is applied only on very basic object forms like cubes, spheres, and pyramids. --This work lacks in maturity required for Autonomous Research.
Domain Randomization and Pyramid	--This paper proposes to harness the potential of simulation for the semantic segmentation of real-	--The generated dataset is evaluated only for semantic segmentation.

Consistency: Simulation-to-Real Generalization without Accessing Target Domain Data, by X. Yue, Y. Zhang, S. Zhao, A. Sangiovanni-Vincentelli, K. Keutzer, B. Gong, 2019	world driving scenes in a Domain Randomization fashion. --They randomize the synthetic images with auxiliary datasets and enforce pyramid consistency across domains within an image.	--It would be important to establish how this dataset performs for other general-purpose Computer Vision tasks as well.

Table 2: Related works

2.7 Thesis Statements

2.7.1 Problem Statement

Machine learning based Autonomous Vehicle technology requires a colossal volume of data, encompassing a variety of driving scenarios and conditions, to be trained upon. Simulation environments have emerged as an efficient, safe and cost-effective solution for the training, testing, and validation of Autonomous Vehicle technology. However, for the effective usage of simulation environments, the complication of Reality Gap (difficulty of transferring simulated experience into real-world) must be addressed. In this work, the usage of highly realistic 3D models of dynamic objects (traffic actors) in pre-built city models enables the simulation environment to look highly photo-realistic. In this manner, the data generated will be quite realistic in terms of photo-realism enhancing the training efficiency of Deep Learning based Computer Vision techniques. Also, the flexibility and ease at which scenarios can be tailored using our data generation tool will enable efficient, safe, fast and cost-effective data collection for Autonomous Vehicle research.

2.7.2 Thesis Contribution

The work of this thesis can be divided into two parts:

A) The work implemented as a part of the overall system:

- Set up a realistic simulation environment with pre-built 3D city models and 3D models of dynamic objects (provided within the simulator being used).
- Using prior knowledge about the driving scenario (traffic actor classes, their pose, location, speed, etc.), the scene is recreated in the simulation environment.
- Based on that data, perform Motion Prediction and Risk Assessment and determine the traffic actors that pose a threat to the ego vehicle.

B) **Main contribution** of this thesis:

- The original scene is then altered (by changing various parameters like the number of traffic actors, their original positions, trajectories, pose, etc.), and various test scenarios are generated.
- In this manner, the main contribution of this thesis would be a test scenario generation environment (tool) that allows a user to define desirable driving scenarios for data collection for training and testing Autonomous Vehicle algorithms.
- This environment (tool) was set up using CARLA, an open-source simulator, and various Python scripts were written to control the simulation environment (some scenarios were explicitly defined while some were created in a random fashion).
- The other main contribution will be an annotated dataset (framewise collection of the generated test scenarios) that was created as a result of the implementation part of this thesis.
- This dataset can be readily used for Autonomous Vehicle research. Also, new data can be easily collected using the above-mentioned test scenario generation environment.

Chapter 3: Proposed System

This chapter first discusses how this work is related to the overall system developed by a group of six students under supervisor Dr. Xiaobu Yuan, University of Windsor. Secondly, this chapter discusses the proposed system developed for “test scenario generation” for testing and validation of Autonomous Vehicle algorithms.

3.1 Motivation

Recently, test Autonomous Vehicles of top companies like Uber and Tesla caused pedestrian fatalities [62], raising safety concerns. Such incidents point to the fact that Autonomous Vehicle algorithms still need to be trained on vast amounts of data before being deployed on roads for consumer use. The proposed system is developed with an aim to generate huge amounts of realistic data for Autonomous Driving research in a safe, time-saving and cost-effective manner.

3.2 Working of the overall system (developed by a group of six students)

The overall system is composed of six components as follows:

- Construction of a virtual 3D environment
- Rendered images of real-time video
- 3D feature and keypoint extraction
- Removal of static and variable objects
- Dynamic object recognition
- Simulation Environment

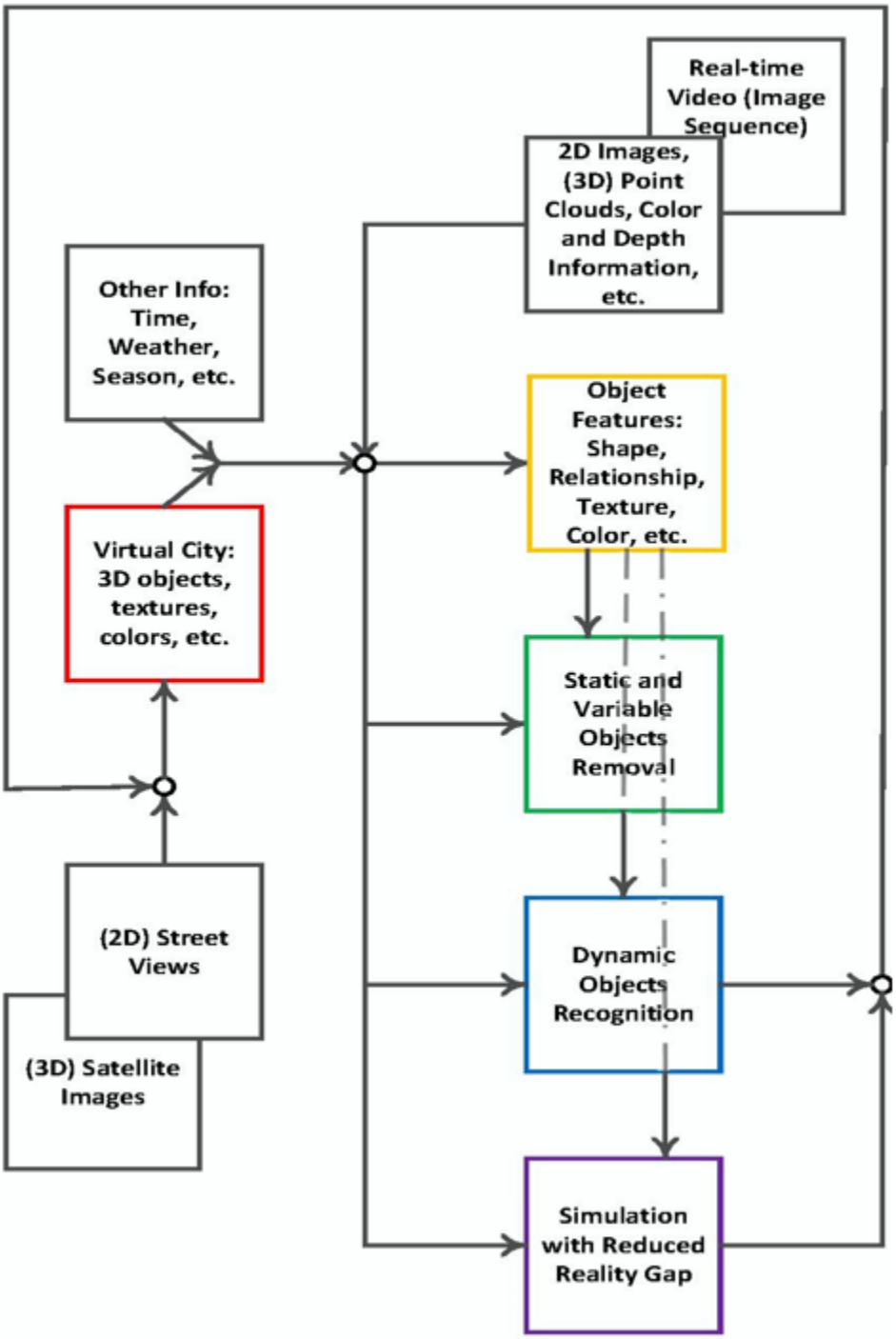

Figure 22: The overall System architecture

The above figure describes the architecture of the overall system (developed by six students) and how different modules are connected to each other. The first part (red) deals with the construction

of a virtual 3D environment using OpenStreetMap data (VGI/crowdsourced) and façade texture from Google street view images. The virtual 3D city model contains stationary objects such as buildings and variable objects like trees. After that, several features are extracted (yellow) from the 3D object models and stored in a repository. This repository is then further used in the Dynamic object recognition task. The module marked in green deals with stationary and variable object elimination, where the key points are first detected in the input image to verify the existence of that object in the real-world by matching the extracted keypoints of the input image. Matching the features of the virtual environment and real-time image confirms the location of the car in the real-world that solves the problem of geo-localization of a self-driving car. After that, dynamic object recognition and pose estimation (blue) are carried out identifying the object class, its location, pose and speed.

This information is then passed over to my part (Simulation Environment - violet). I have to localize the dynamic objects and recreate the scenario in the simulation environment. Furthermore, I perform motion estimation and risk assessment and determine which objects can be a threat to the ego vehicle.

This concludes my role in the overall system. Apart from that, the main focus of this thesis is test scenario generation which will be discussed in detail later on.

From the above-mentioned six modules, the ones from which I receive my inputs are discussed in brief as follows:

- Construction of 3D Virtual World: Firstly, a virtual city is constructed using open source VGI data such as 2D street views and satellite images. 3D structural files are extracted with 3D structures of the buildings that are rendered, and the final 3D structure is obtained with

the geolocation information that is externally mapped on to the model. Textures are mapped onto buildings in the 3D model by extracting real-world images and georeferencing them. In this way, a virtual city with stationary (e.g. buildings) and variable objects (e.g. trees) is formed. Later this virtual city is updated with dynamic objects using real-time recognized dynamic object details. The virtual city with 3D static, variable, and dynamic object model information present in real-time road scenes is used by the self-driving car to navigate safely by knowing the surroundings. This module is marked red in figure 22.

- Dynamic Object Recognition: This module matches features of the dynamic objects in the input image with the feature information of 3D object models stored in the repository to find a suitable match of 3D model for each of the dynamic objects present in the input image. After finding the corresponding 3D model from the repository, a voting algorithm is used for the matching purpose, and to estimate the confidence score that signifies the assurance of object identification. This process improves the confidence of recognition and pose estimation of dynamic objects in the input image. This module is marked in blue in figure 22.

3.3 Proposed System Architecture

This section discusses the architecture of the system that comprises of all the work done in this thesis.

3.3.1 Main System Architecture

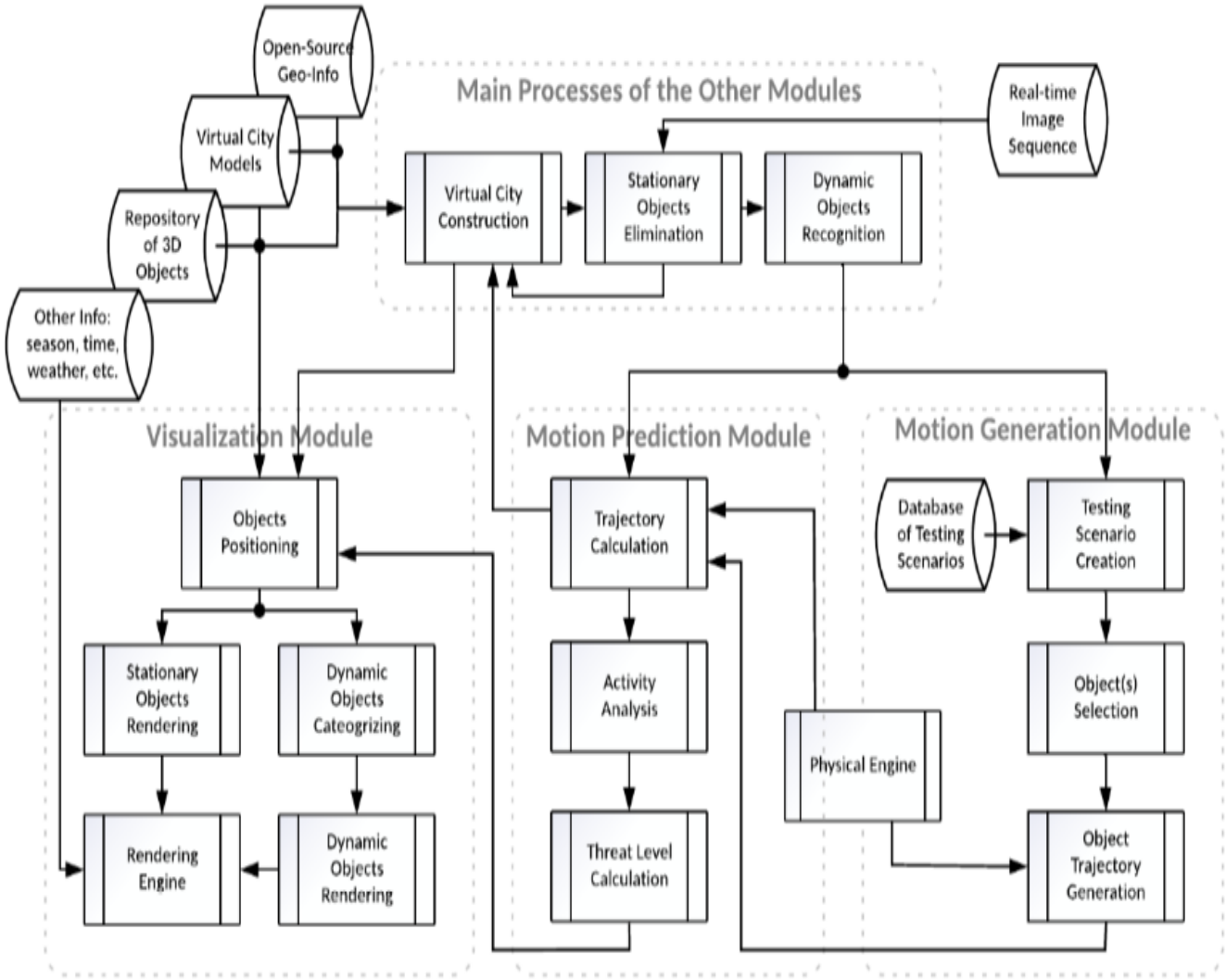

Figure 23: Proposed System Architecture

As shown in the above figure, my system consists of three main modules: The Visualization module, Motion Prediction and Risk Assessment module, and the Motion generation (test scenario generation) module.

- Visualization Module: As the name suggests, this module hosts the simulation side of the system. Based on the prior information (object class, its pose, location and speed), objects are localized accordingly in the simulation environment, and a driving scene is initiated.
- Motion Prediction and Risk Assessment module: This module takes in the necessary information (current location, object class and speed) and predicts the future trajectory of the object. Then risk assessment is performed and determined whether an object will be a threat to the ego vehicle or not.
- Motion generation module: Once the original scenario is executed, various parameters like the number of objects, their speeds, orientation, weather and lighting conditions, etc. are altered, and new test scenarios are generated. Such test scenarios are stored in the form of an annotated dataset that can be used for computer vision research. Such test scenarios are generated by explicitly defining the parameters as well as in random fashion.

3.3.1.1 Proposed System Algorithm

Algorithm: Coordinated working of the three modules

Input: Prior scene information, simulator

Output: Reconstructed original scene and its variations with Motion Prediction and Risk Assessment applied

---

Step 1: Receive prior information about the driving scenario (number and type of actors, their pose, speed, etc.) from other processes.

Step 2: Based on this information, the original scenario is recreated in the Visualization Module.

Step 3: Optional: Apply Motion prediction and Risk Assessment module to this scene.

Step 4: Feed this scene to the Motion Generation module and construct many test variants of the original scene by changing relevant configurations and store them in the desired data format.

Step 5: Apply the Motion Prediction and Risk Assessment module to each of these test scenarios to validate them.

Step 6: Reflect the Motion prediction and Risk Assessment results visually by highlighting the actors by color code (Red for high risk and Yellow for mild risk).

Step 7: Repeat steps 1-3 for a new original scenario.

Step 8: Repeat steps 4-6 to generate test scenarios again out of this original scenario.

Step 9: Exit.

3.3.2 Visualization Module

The simulation environment is hosted on the CARLA 0.9.5 simulator. The simulator is discussed in detail in section 2.4.1. In the main system, real-time dynamic information will be provided as prior information to our Visualization Module through the IoT architecture. The prior information contains all the scene information like the landscape, number of objects, object classes, orientation, static and variable object information, weather information, etc. For our experimentation, we assume the prior information to initiate the scene. The visualization module architecture is as shown in the figure below.

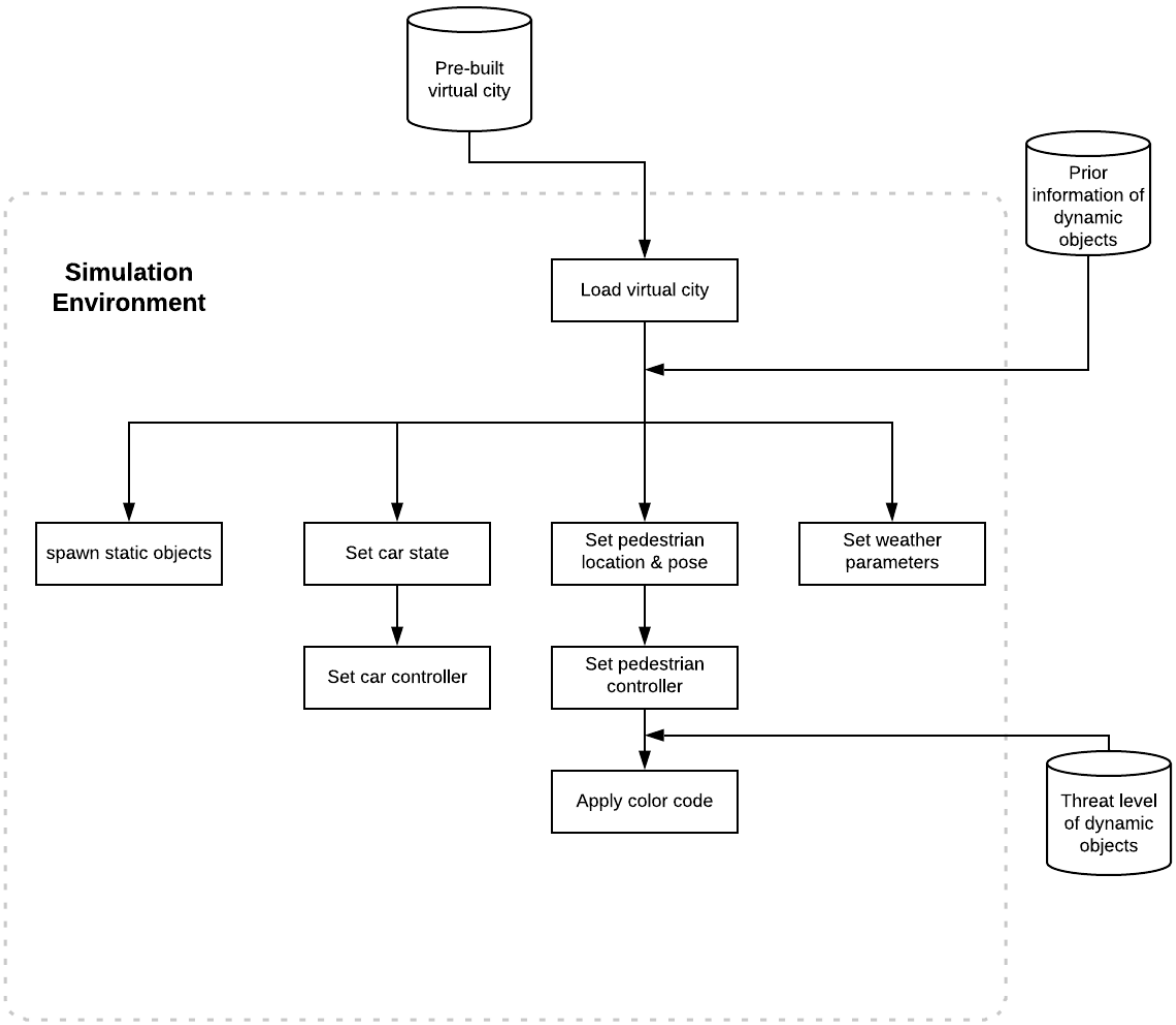

Figure 24: Visualization Module Architecture

The step-by-step explanation of the process of generating a scene in the simulation environment is as follows:

- Loading Virtual city: For this experimentation, the virtual city used is Town03, provided in the simulation environment itself. This is because the spawn points and other parameters are hard-coded in the provided environments. We can expect that the driving simulator conveniently allows using the custom environment in its further stable releases.

- Spawn and Activate actors: Once the landscape is loaded, the actors, (vehicles, humans, and other static/variable objects) need to be spawned into the simulation environment at prior known locations and orientations.
- Define and assign controllers for actors: Once the actors are spawned in the environment, the next step is to define and assign controllers to them to make them move in a desired manner. In this step, we can define various controller parameters depending on the actor type. For vehicles, the parameters that can be defined are throttle, steer, brake, hand_brake, reverse, etc. The parameters for pedestrian type actors are speed, rotation, heading, direction, etc. Apart from custom-defined controllers, we can also assign AutoPilot modes to dynamic actors. However, there are no such controller options for static objects in the scene.
- Define Weather conditions: Finally, the weather conditions such as cloudiness, precipitation, sun_altitude_angle, sun_azimuth, etc. are defined as per the prior information.
- Finally, depending on the Motion Prediction and Risk Assessment results, the respective objects are highlighted with specific color codes.

3.3.2.1 Visualization Module Algorithm

Algorithm: Recreation of an original scene in the Visualization module based on prior information

---

Input: Prior information and simulator

Output: Recreated scenario in the simulation environment

Step 1: Based on the prior information, activate the correct number and type of actors.

Step 2: Spawn the actors (vehicles, pedestrians and static) at designated locations.

Step 3: Configure the vehicle controllers and walker controllers by mainly setting the designated velocity and orientation, among other parameters.

Step 4: Define the required weather parameters such as cloudiness, precipitation, and sun_altitude_angle.

Step 5: Spawn and configure the required photographic, Lidar, or radar sensors.

Step 6: Attach the sensors to required actors to capture data from different points of view.

Step 7: Set a timeout after which all the actors in the simulation should get destroyed.

Step 8: Exit.

3.3.3 Motion Prediction and Risk Assessment Module

Once, the original scene has been initiated, Motion Prediction and Risk Assessment are performed on the nearby dynamic actors, w.r.t the ego vehicle. The dynamic objects that can be a threat to the ego vehicle are identified and are highlighted with color codes.

It is important to note that we have used the work of F. Chan et. al [63] in this module. This work was used as it is for our data validation and **no contributions** were made to it. F. Chan et al. contributed a Dynamic-Spatial-Attention (DSA) Recurrent Neural Network (RNN) for anticipating accidents in dashcam videos. Their method anticipates collisions about 2 seconds before they occur with 80% recall and 56.14% precision.

While extensive discussion about their approach is out of the scope of this thesis, the risk metric used in their work is Time to Accident. We will call this risk metric as Time to Collision in this thesis. The pre-trained collision anticipation model implemented by F. Chan et al. can be found at [64]. While this model is a demo model, we trained our model on the licensed dataset provided by

the authors upon requesting it. This dataset consists of 678 dashcam videos on the web. The dataset is unique since various accidents (e.g. Motorbike hits a car, a car hits another car, etc.) occur in all videos. This dataset is quite similar to the simulated data we generated in terms of collisions. Hence, we could easily apply their model on our test data to extract just the Time to Collision for colliding objects.

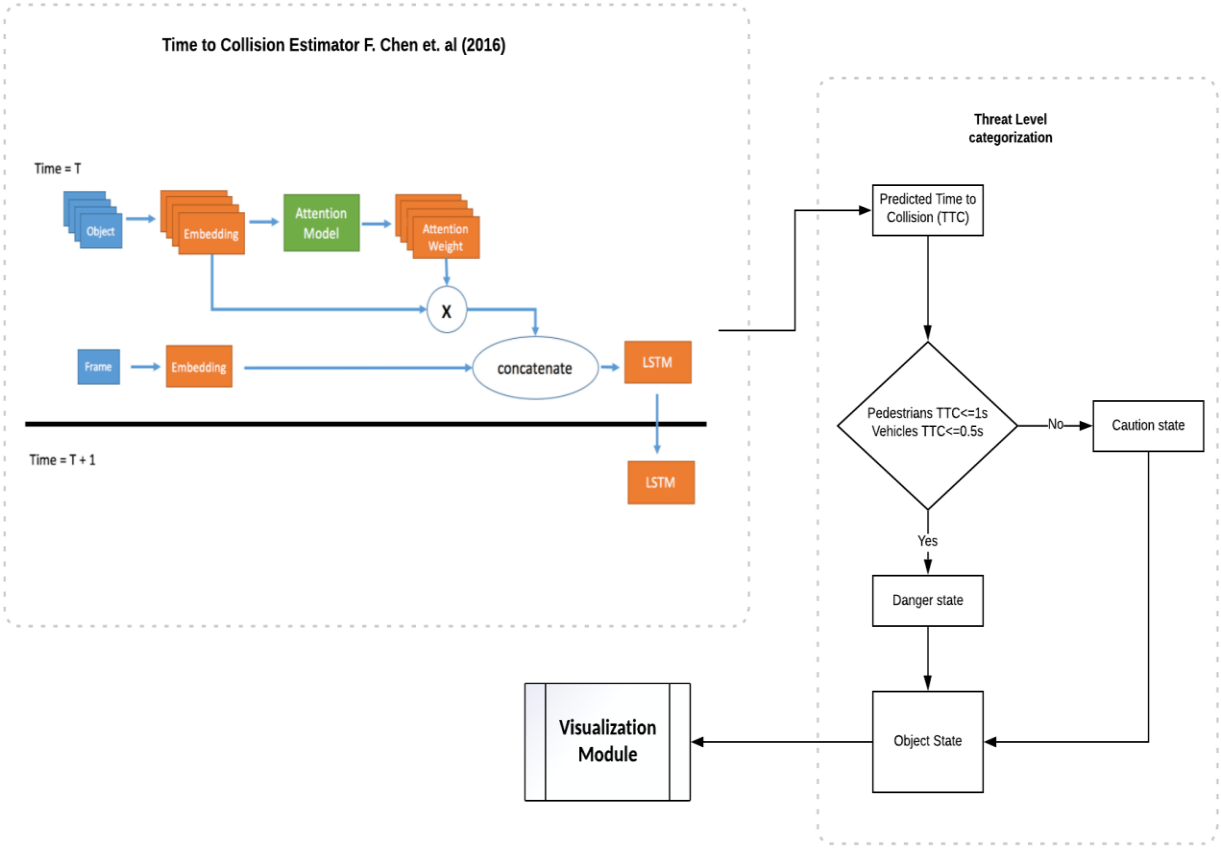

Figure 25: Motion Prediction and Risk Assessment Architecture

As shown in the above figure is the Motion Prediction and Risk Assessment architecture used in this work. The DSA RNN model by F.Chan et. al. forms the core of this module and calculates the Time to Collision (TTC) for colliding objects only. We further categorize objects as per their TTC.

For pedestrians the state changes to “Danger” when the TTC is less than or equal to 1 second. For vehicles, the state changes to “Danger” when the TTC is less than or equal to 0.5 seconds. For objects who do not collide into the ego vehicle at all, this model doesn’t calculate TTC. So we have manually categorized them as in “Caution” state once they enter the intersection area used in this experimentation as described in section 4.2. While the objects who are not detected at all by the object detector (and hence far enough) are not assigned any state and can be considered “safe”. It is important here to note that the thresholds of different actors (TTC<=1s for pedestrians and TTC<=0.5s for vehicles) has been manually fixed to indicate when they should be considered a danger to the ego vehicle. The Time to Collision itself is calculated considering parameters like the actor type, current location, orientation and velocities of those actors w.r.t the ego vehicle.These parameters do not affect the threshold values for categorizing the dynamic actors into threat states as the TTC is calculated first based on these parameters and the threshold is just used here to divide the TTC range.

3.3.3.1 Motion Prediction and Risk Assessment Algorithm

Algorithm: Time to Collision Estimation and assigning Threat state

---

Input: Image frames

Output: Threat state of colliding object w.r.t ego vehicle

Step 1: Apply the collision estimation model by F. Chan et. al (2016) on the required image frames.

Step 2: This model will provide the Time to Collision of any objects that collide into the ego vehicle.

Step 3: For pedestrians, assign the “Danger” state when TTC<=1s.

Step 4: For vehicles, assign the “Danger” state when TTC<=0.5s.

Step 5: For other objects within a pre-defined distance (within the intersection for this experimentation), assign threat state “Caution” manually.

Step 6: For objects not detected by the object detector (and hence far enough), they can be considered “safe”.

Step 7: Forward the object threat state information to the Visualization module.

Step 8: Exit

Apart from [63], there are other recent works such as [65], [66], [67], [68], and [69] that can be referred for Dynamic Risk Assessment in traffic scenes. In fact, [65] and [69] categorize the objects elaborately into categories like low risk, mild risk and high risk, which is exactly what we wanted to do on our test scenarios. However, we could not procure their code and datasets for our usage. As a result, the model from [63] was used finally.

3.3.4 Motion Generation (Test Scenario Generation) Module

Once the original scene has been executed successfully, the Motion Generation module can be employed to alter the scene and generate various challenging situations out of the original scene. The Motion Generation module architecture is shown in the below figure.

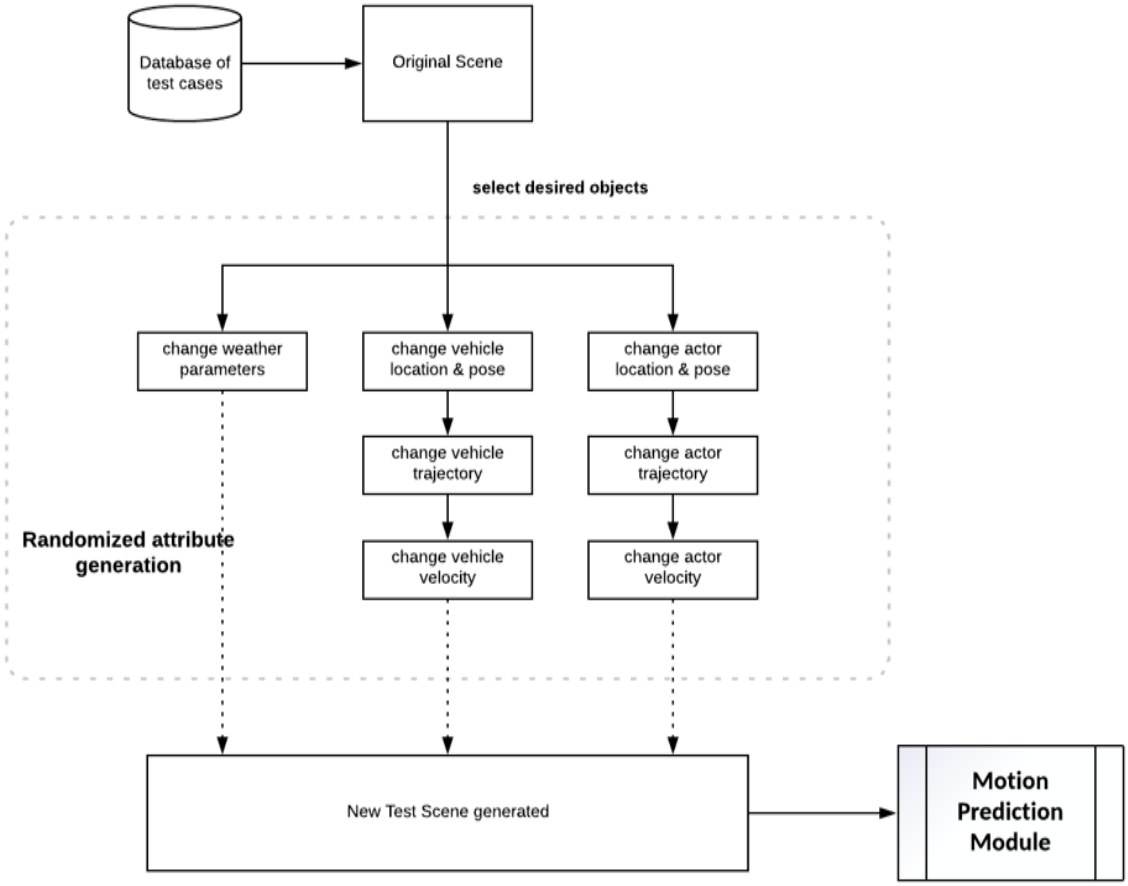

Figure 26: Motion Generation Module

As shown in the above figure, first of all, the original scene is loaded in the simulation environment. After that, the tester has the choice to select the desired number/models of vehicles, pedestrians, static objects, etc. and spawn them into the scene at desired locations. The tester can also change some generic features like vehicle colors and types and the weather type. Finally, the tester can define the motion (speed, orientation, direction) that a particular actor will follow. In this manner, simply by adjusting certain configurations in the original scene, a new test scene can be generated. Furthermore, motion prediction and Risk assessment algorithms can be applied to this newly constructed scene and the results can be observed.

In this work, various behavior-based movement patterns have been defined and stored as various python scripts. The movement patterns implemented in this work are,

- a pedestrian moving on a sidewalk at less velocity
- a pedestrian moving across the intersection in a dangerous manner,
- a pedestrian running into the ego vehicle,
- a car running into the ego vehicle from across,
- a car running into the ego vehicle from left,
- a car taking a dangerous turn near the ego vehicle
- two cars colliding in front of the ego vehicle
- a bike running into the ego vehicle from right, etc.

Further on, the above-discussed behaviors can be mixed and matched along with the number of actors, types, weather conditions, etc. and various other scenarios can be constructed at ease. The parameters and attributes responsible for generating various movement patterns have already been discussed in section 3.3.2. These parameters need to be adjusted ***relative*** to the current motion state of the ego vehicle to make it move relative to the ego vehicle.

3.3.4.1 Test Scenario Generation Algorithm

Algorithm: Generating test scenarios by altering the original scenario

---

Input: The original scenario generated in section 3.3.2

Output: Multiple test scenarios

Step 1: Select and adjust the number and type of actors from the original scenario.

Step 2: Spawn the actors (vehicles, pedestrian and static) with color variations at desired locations.

Step 3: Change the vehicle controller and walker controller configurations to create desired movements relative to the ego vehicle.

Step 4: Change the weather parameters to achieve the required variations in weather conditions.

Step 5: Set the location at which the recorded data will be stored.

Step 6: Set a timeout after which all the actors in the scene will be destroyed.

Step 7: Run the scenario. New test scenario executed and recorded.

Step 8: Repeat steps 1-7 as many times with relevant variations to generate as many test scenarios.

Step 9: Exit.

Chapter 4: Results and Experimentation

The proposed system was built on a PC running Windows 10, with Intel(R) Core(TM) i7-5820 processor, NVIDIA GeForce GTX 960 GPU, and 32 GB of RAM. The list of software, libraries and tools used during this implementation is provided in the table below.

CATEGORY	NAME
Operating System	Windows 10
Programming Language	Python 3.7.x, Python 3.5.x
Rendering Software	Unreal Engine 4.18
Driving Simulator	CARLA 0.9.5
IDE	Jupyter Notebook, Anaconda, Sublime Text
Libraries	OpenCV, NumPy, CUDA toolkit, Tensorflow, random, time, etc.

Table 3: Software, Tools and Libraries

4.1 Simulation Environment and User Interface

As discussed earlier, the driving simulator used for this experimentation is the CARLA 0.9.5 (Windows version). The python client scripts to control the simulation environment were written in Python 3.7. Some screenshots demonstrating the simulation environment are as follows:

Figure 27: CARLA logo

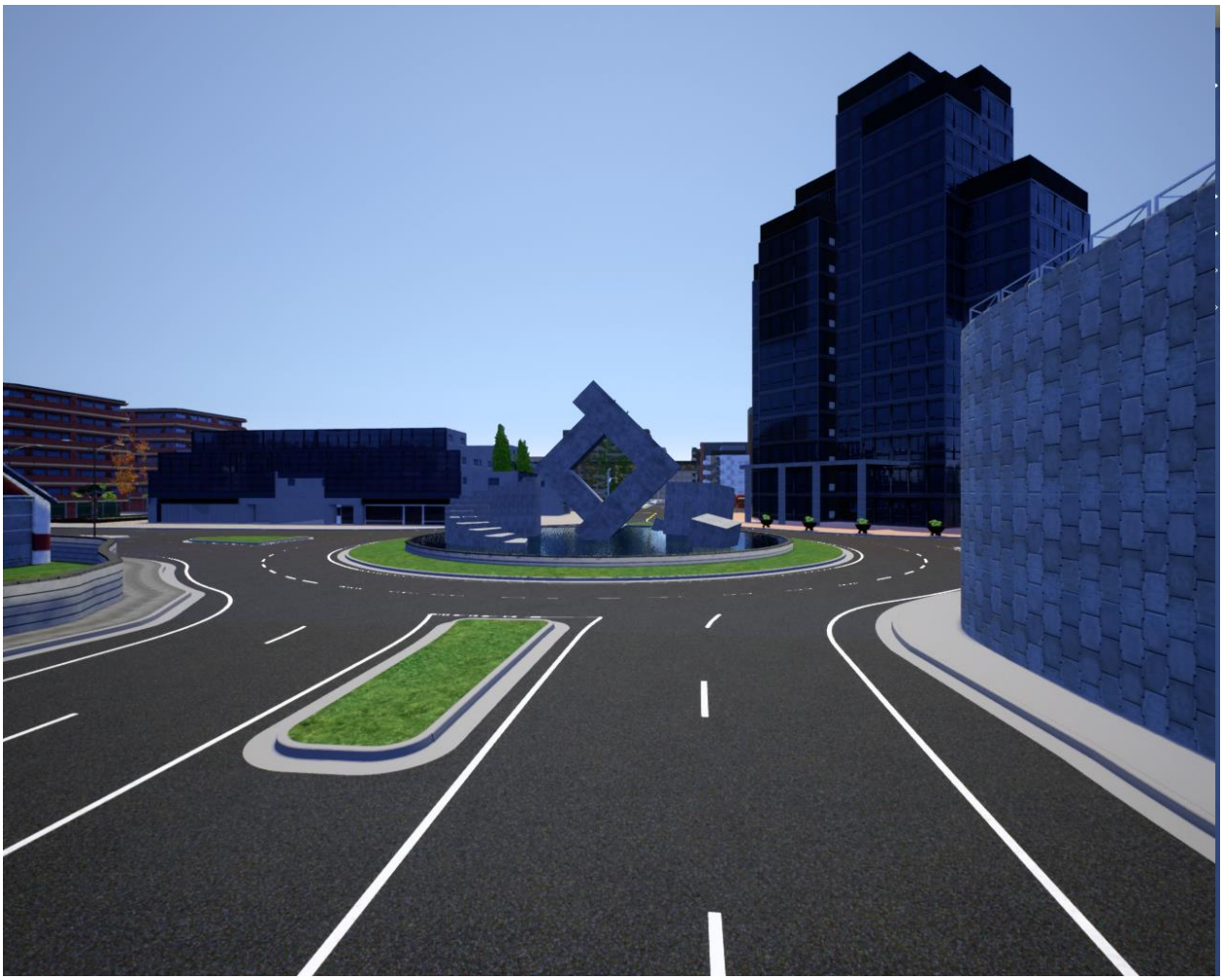

Figure 28: Sample scene from CARLA simulator

4.1.1 User Interface

In order to make our simulation environment user-friendly, we built a basic interface using which a user can run scenarios or edit them without actually having to deal with the complicated python scripts. The interface consists only of those functionalities that were actually built and tested in this work and does not have control over the rest of the functionalities of the simulator. The flowcharts describing the flow, navigation and tasks are as follows.

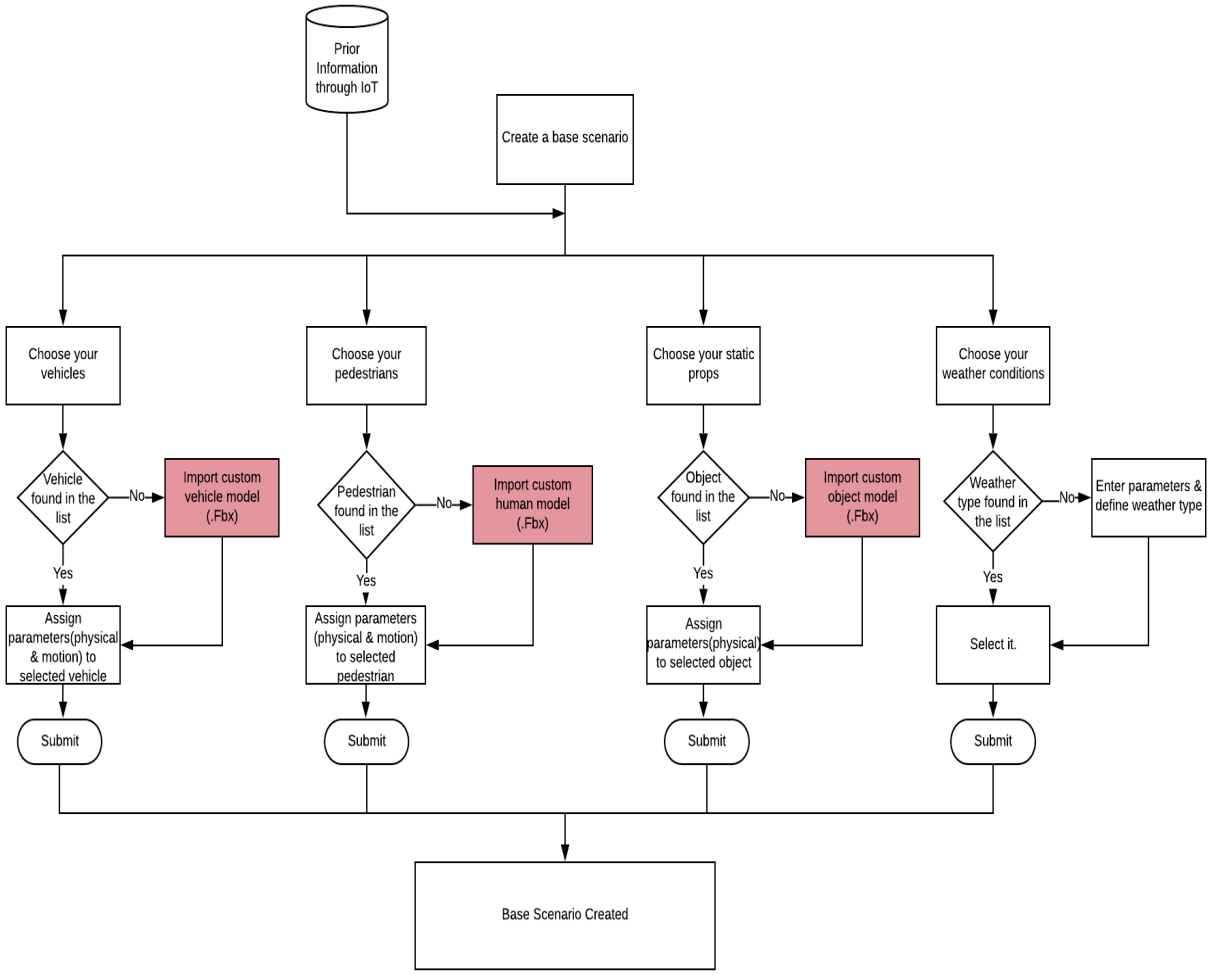

Figure 29: User Interface: Base Scenario Creation

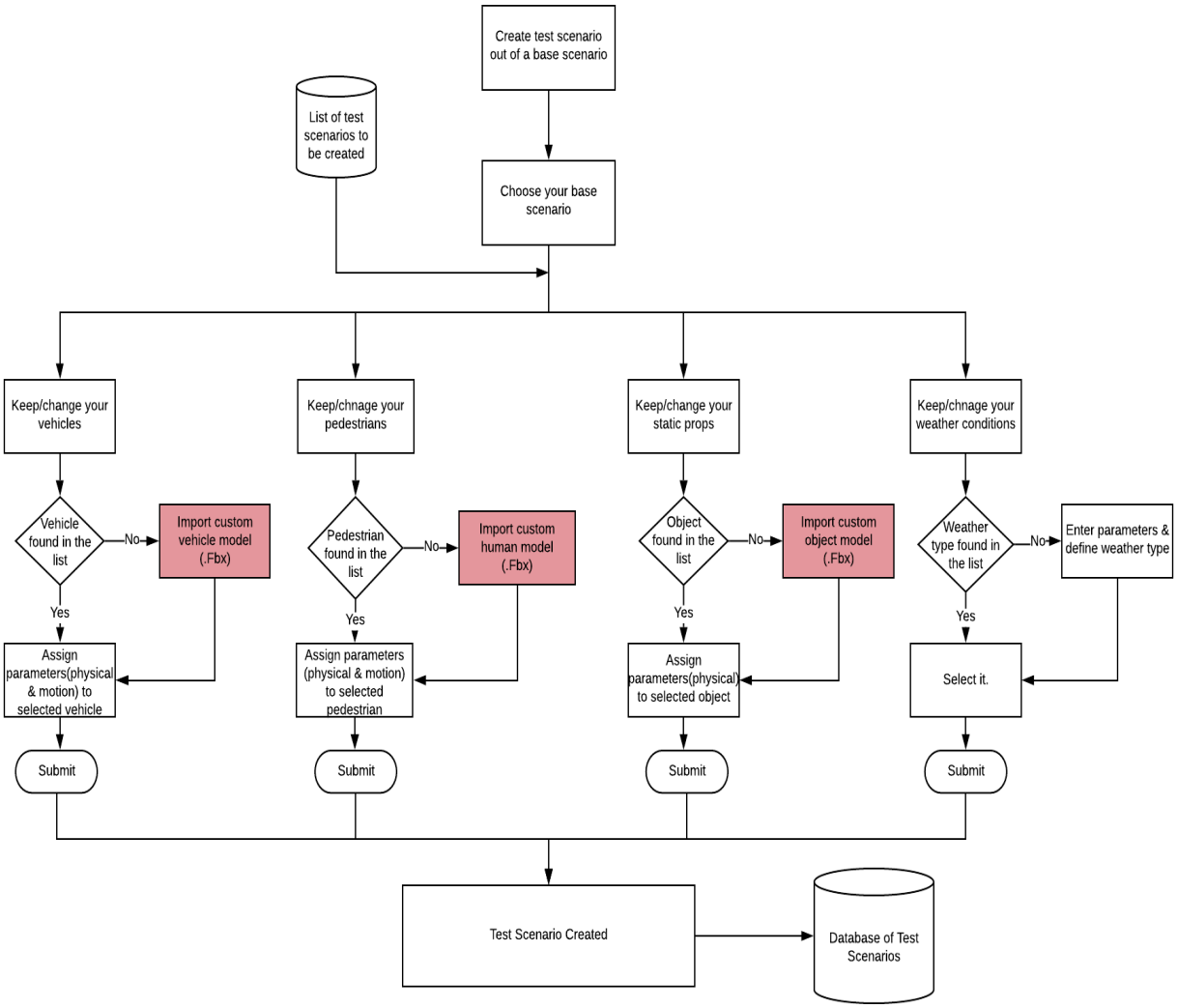

Figure 30: User Interface: Test Scenario creation

The above-displayed flowcharts are quite similar to the flowcharts described in sections 3.3.2 and 3.3.4 as they represent the same processes (base scenario generation and test scenario generation). The screenshots describing the actual interface are as follows:

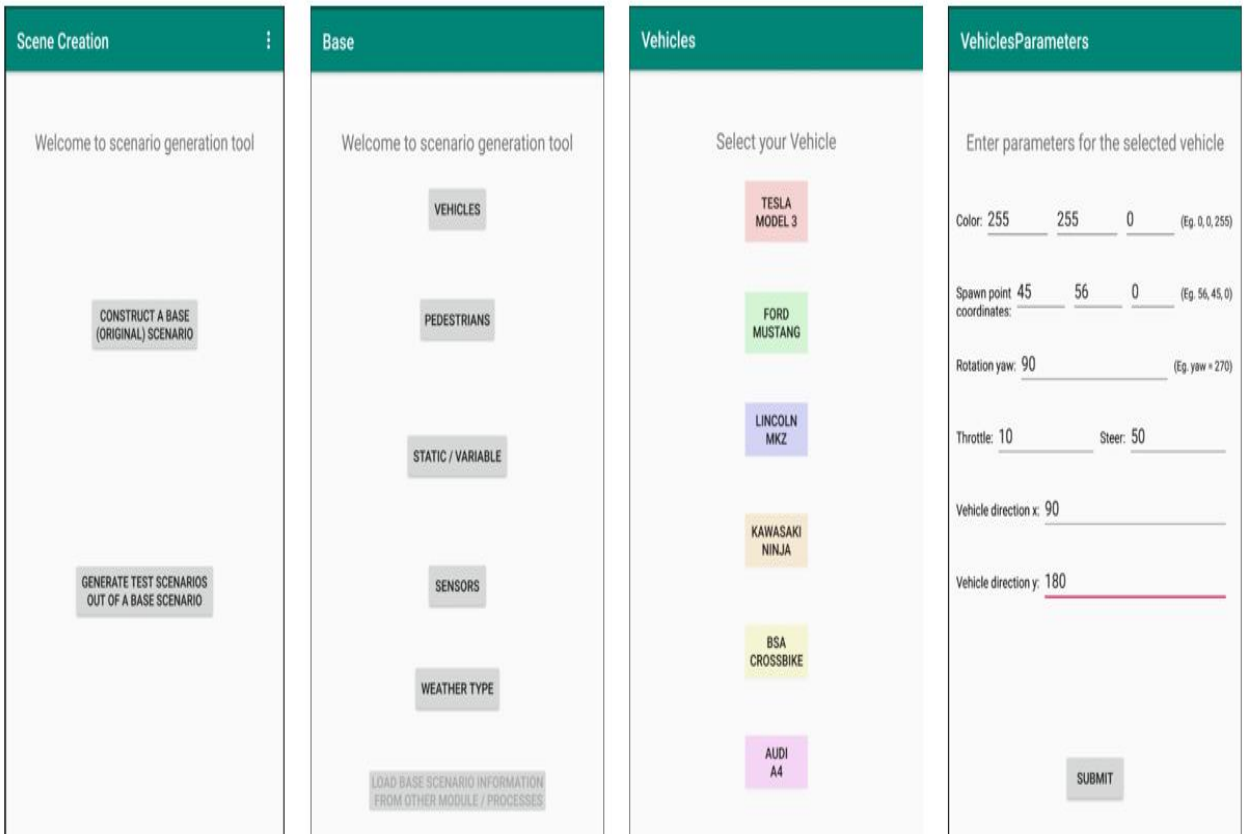

Figure 31: User Interface screenshots (a)

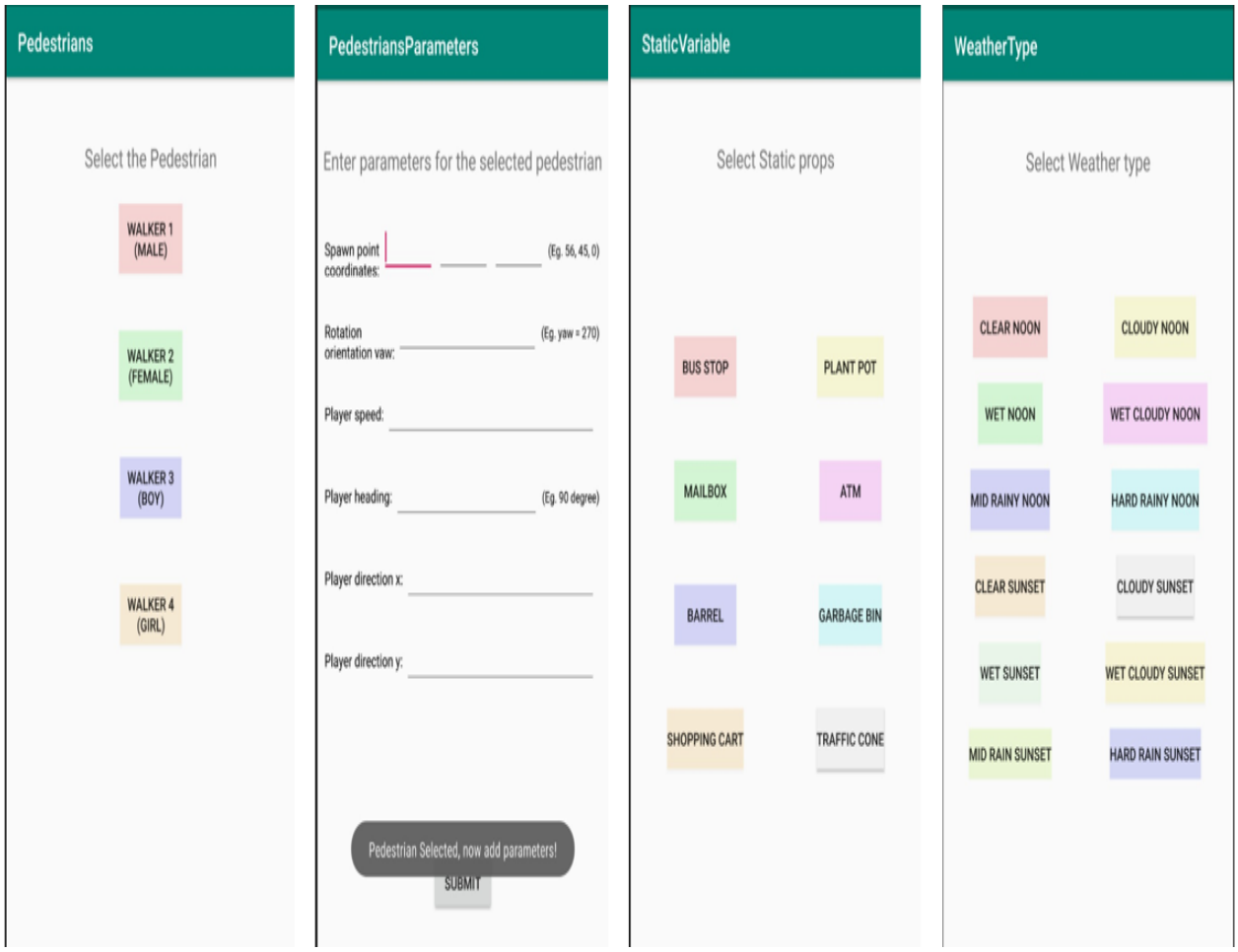

Figure 32: User Interface screenshots (b)

4.2 Driving Scenario Results and Specifics

This section discusses the driving scenarios implemented using the scenario generation tool developed in this work.

It is important to note that the simulator used currently doesn't allow us to import and use custom-built 3D object/landscape models, and hence, we had to use the objects/landscapes provided within the simulator. This is because the locations, objects, waypoints for their relative movements, etc.

are hardcoded within the simulator and the best results can be gained only by using the way they are supposed to be used. If we import custom-built 3D object/landscape models, the driving functionality (the soul of Autonomous Driving research) of the simulator would not work efficiently.

In this manner, the base scenario created in this experimentation is as realistic as possible in terms of relative positions of actors, their relative movements, placement of static props, right time, season and weather of the day, and the photorealism of data captured by the sensors.

In this experimentation, we have provided a **proof of concept** showing how we can use the prior information (including the right number of static, stationary, and dynamic objects in the right time, season, and weather of the day) and recreate a scene with the best resources available currently.

Further releases of Driving Simulators will get mature and allow us to use custom 3D objects, but as of now, we have exploited the thorough potential of the best open-source Driving Simulator (CARLA) in terms of scenario generation.

Furthermore, **the functional simulation environment and the scenarios generated are the main results of this work**. Apart from that, the objects in the frames are highlighted by color codes Red (Danger) and Yellow (Caution) as per their threat levels w.r.t the ego vehicle. The risk assessment model used in this work is from [63]. This model was just applied to our scenarios without making any changes and **we are not claiming any contribution to the Motion Prediction and Risk Assessment done in our work.**

We can say that we have validated our generated data using Motion Prediction and Risk Assessment techniques. In similar works like [56], [57], [58], [59], etc. where synthetic data is generated it is validated by applying various Computer Vision techniques like Object Recognition

and Object Detection. Here validation of the data simply means that Computer Vision techniques were applied on the generated data as well as other standard datasets and then the results are compared, which indicates the quality of data. However, such tasks are usually carried out by big teams as it is time consuming as well as labor-intensive to manually label the huge amount of generated data, preprocess it and then apply Computer Vision models on it to get the results. So, in our work we simply applied the DSA RNN model for anticipating collisions provided by F. Chan et. al. [63] on our test data and were able to obtain genuine results in the form of TTC which proves that our data can be readily used. This DSA RNN model includes Object Recognition as one of its intermediate steps as well.

In section 4.3, for each image, we have provided a table detailing the raw data (objects, their positions, directions, Time to Collision and threat state w.r.t the ego vehicle). The Motion Prediction and Risk Assessment model used is an end-to-end model (doesn't involve any significant intermediate steps) and detects collisions only (either an object can be safe or dangerous enough to collide into the ego vehicle). The dangerous objects in our scenarios are highlighted Red along as per their Time to Accident (Time to Collision - <=0.5s for vehicles and <=1s for pedestrians) computed using [63]. However, in this work, we also have manually categorized the objects within the intersection into "Yellow(Caution)" for demonstration purposes. While the objects which were not detected by the Object Detector (far enough and hence safe from the ego vehicle) are not highlighted at all.

4.2.1 Original Scenario

As discussed in section 3.3.2, the original scene is initiated based on the prior information received from other modules through the IoT. For our experimentation, we assume this prior information about the number of objects, type of objects, location, velocity, orientation, etc. such that it can

resemble the properties of a real traffic scene. The original scene is based at a clear intersection in Town 3 (shown in the figure below) provided by the Carla simulator.

Figure 33: Selected Intersection for Experimentation

The set of actors used in this implementation is as follows:

- Vehicles: A Tesla Model 3, a Ford Mustang, Lincoln mkz 2017 and, a Kawasaki Ninja were the vehicles used in the original scene.
- Pedestrians: The two pedestrian types used were “0001” and “0002” provided withing the simulator. It is expected that CARLA will provide a diverse variety of pedestrians in their upcoming stable releases.
- Static Props: Various static objects like shop, bus stop, atm, mailbox, etc are placed in the scene in order to resemble a realistic environment.

4.2.1.1 Initial positions of dynamic objects

The below figure shows the initial positioning of dynamic objects around the intersection.

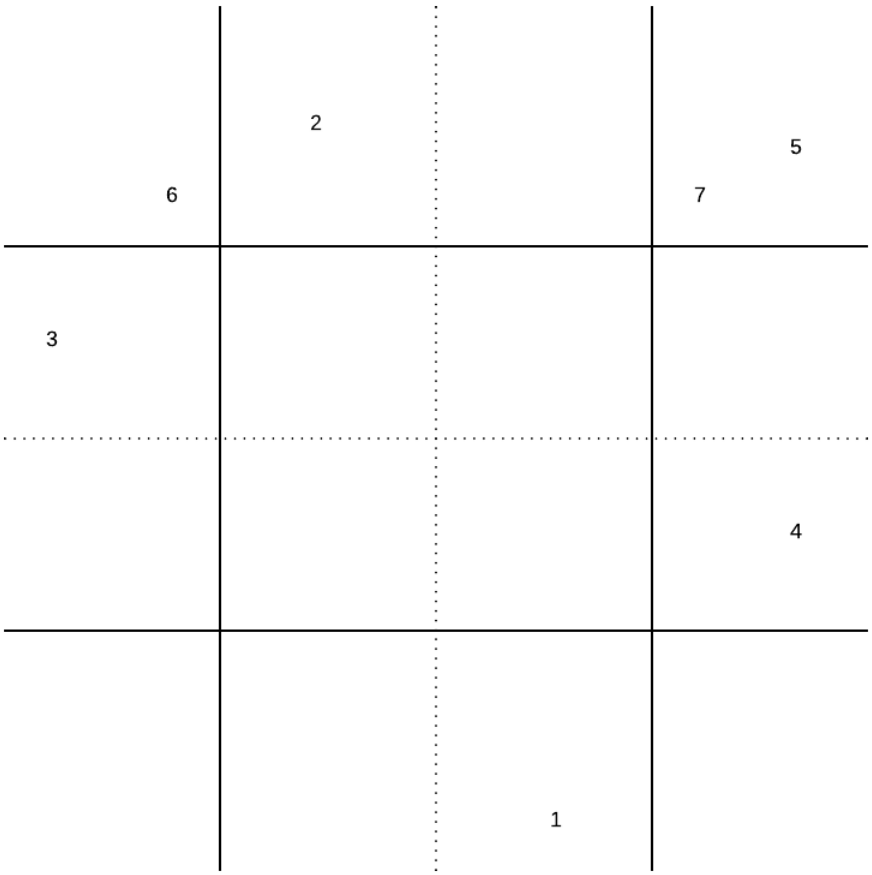

Figure 34: Initial Positions of Dynamic Actors

The actors corresponding to each number are as follows:

- 1: Tesla Model 3 (Red) (**Ego Vehicle**)
- 2: Ford Mustang (Green)
- 3: Lincoln mkz 2017 (Blue)
- 4: Kawasaki Ninja (Blue)

- 5: Pedestrian type 0001
- 6: Pedestrian type 0002
- 7: Pedestrian type 0001

4.2.1.2 Original Scenario execution

In the original scenario, each actor is allowed to execute its natural movement pattern. There is no collision or dangerous maneuvers in the original scenario. In this scene, the Red car is our ego vehicle. The scene from the ego car's point of view is shown in the below image sequences.

Figure 35: Base Scenario, Image 1

Figure 36: Base Scenario, Image 2

Figure 37: Base Scenario, Image 3

Figure 38: Base Scenario, Image 4

4.3 Test Scenario Generation along with Motion Prediction and Risk Assessment

This section includes the results demonstrating how the original scenario was altered by adjusting various configurations and dangerous scenarios were built w.r.t the ego vehicle. Moreover, at a certain point in every test scene, Motion Prediction and Risk Assessment are performed over nearby actors and the potential threats are identified and highlighted using color codes as discussed in section 3.3.4. Keeping the original scene as reference, the different test scenarios generated and validated in this work as discussed as follows.

4.3.1 Test Scenario 1: Blue Car taking a dangerous turn and colliding into the ego vehicle

In this scenario, the Blue car's motion is manipulated and is made to take a turn and run into the ego vehicle. It can be noticed how the Blue car is in the yellow highlight when it is within the

intersection and as it comes dangerously close, it is highlighted in red. This scenario is briefly demonstrated in the image sequence below.

Figure 39: Test Scenario 1, Image 1

Figure 40: Test Scenario 1, Image 2

Object	Location	Direction	State w.r.t ego vehicle	Time to Collision
Ego Vehicle	**(-54.9, 129.0)**	**-178 SW**	**NA**	**NA**
Blue car	**(-78.5, 142.0)**	**-53 NW**	**Caution**	**0.8s**
Ninja bike	**(-74.4, 117.6)**	**88 NE**	**Caution**	**NA**

Table 4: Raw Data - test scenario 1, image 2

Figure 41: Test Scenario 1, Image 3

Object	Location	Direction	State w.r.t ego vehicle	Time to Collision
Ego Vehicle	**(-58.9, 128.9)**	**-178 SW**	**NA**	**NA**
Blue car	**(-70.4, 135.5)**	**-36 NW**	**Danger**	**0.3s**
Ninja bike	**(-74.4, 121.7)**	**88 NE**	**Caution**	**NA**

Table 5: Raw Data - test scenario 1, image 3

Figure 42: Test Scenario 1, Image 4

Object	Location	Direction	State w.r.t ego vehicle	Time to Collision
Ego Vehicle	**(-55.9, 128.9)**	**-178 SW**	**NA**	**NA**
Blue car	**(-67.8, 133.7)**	**-34 NW**	**Danger**	**0.1s**
Ninja bike	**(-74.4,123.5)**	**88 NE**	**Caution**	**NA**

Table 6: Raw Data - test scenario 1, image 4

4.3.2 Test Scenario 2: Jaywalking Pedestrian

In this scenario, a pedestrian's (actor number 6 in figure 34) trajectory is set such that it walks towards the ego vehicle. This scenario is briefly demonstrated in the image sequence below.

Figure 43: Test Scenario 2, Image 1

Figure 44: Test Scenario 2, Image 2

Object	Location	Direction	State w.r.t ego vehicle	Time to Collision
Ego Vehicle	**(-60.6, 129.3)**	**173 SE**	**NA**	**NA**
P1 (leftmost)	**(-87.8, 139.9)**	**-24 NW**	**Caution**	**1.1s**
P2 (rightmost)	**(-89.5, 122.9)**	**-1 NW**	**Caution**	**NA**
P3 (center)	**(-92.3,128.1)**	**69 NE**	**Caution**	**NA**

Table 7: Raw Data - test scenario 2, image 2

Figure 45: Test Scenario2, Image 3

Object	Location	Direction	State w.r.t ego vehicle	Time to Collision
Ego Vehicle	**(-60.6, 129.3)**	**173 SE**	**NA**	**NA**
P1 (leftmost)	**(-72.8, 133.4)**	**-24 NW**	**Danger**	**0.4s**
P2 (rightmost)	**(-72.2, 122.7)**	**-1 NW**	**Caution**	**NA**
P3 (center)	**(-90.5, 132.9)**	**69 NE**	**Caution**	**NA**

Table 8: Raw Data - test scenario 2, image 3

4.3.3 Test Scenario 3: Bike running into the ego vehicle

In this scenario, the Kawasaki Ninja bike is made to take an unethical sharp left turn such that it collides into the ego vehicle. This scenario is briefly demonstrated in the image sequence below.

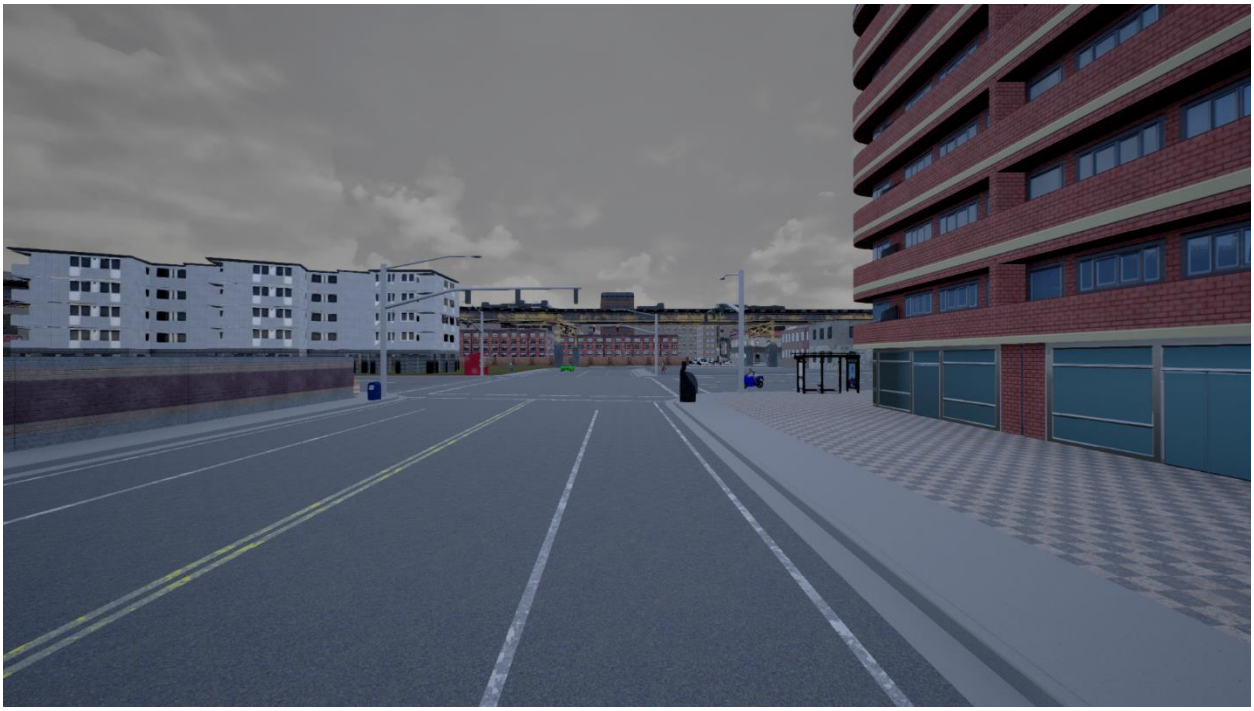

Figure 46: Test Scenario 3, Image 1

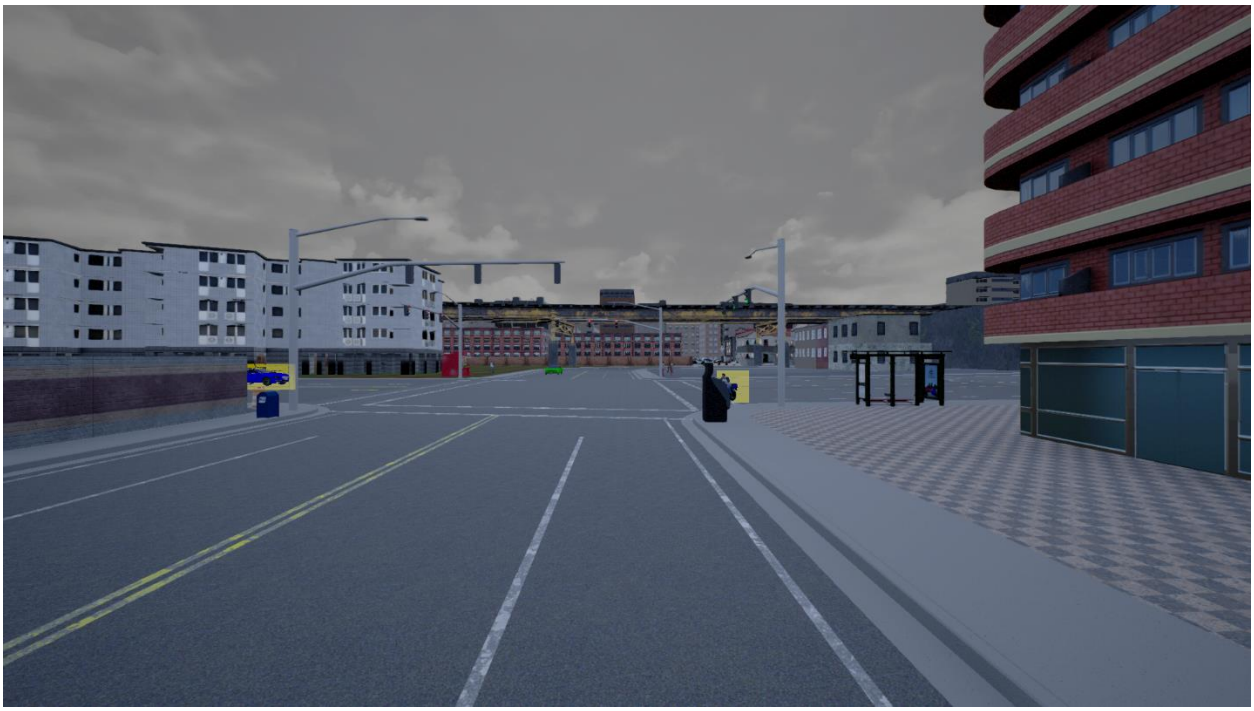

Figure 47: Test Scenario 3, Image 2

Object	Location	Direction	State w.r.t ego vehicle	Time to Collision
Ego Vehicle	**(-35.4, 128.4)**	**179 SE**	**NA**	**NA**
Blue car	**(-85.4, 156.5)**	**-90 W**	**Caution**	**NA**
Ninja bike	**(-73.7, 120.8)**	**78 NE**	**Caution**	**1s**

Table 9: Raw Data - test scenario 3, image 2

Figure 48: Test Scenario 3, Image 3

Object	Location	Direction	State w.r.t ego vehicle	Time to Collision
Ego Vehicle	**(-46.8, 128.7)**	**179 SE**	**NA**	**NA**
Blue car	**(-85.4, 143.7)**	**-90 W**	**Caution**	**NA**
Ninja bike	**(-69.0, 126.0)**	**39 NE**	**Danger**	**0.4s**

Table 10: Raw Data - test scenario 3, image 3

Figure 49: Test Scenario 3, Image 4

Object	Location	Direction	State w.r.t ego vehicle	Time to Collision
Ego Vehicle	**(-57.3, 128.9)**	**179 SE**	**NA**	**NA**
Blue car	**(-85.3, 128.8)**	**-90 W**	**Caution**	**NA**
Ninja bike	**(-62.3, 126.9)**	**8 NE**	**Danger**	**0s**

Table 11: Raw Data - test scenario 3, image 4

4.3.4 Test Scenario 4: Car running into the ego vehicle from across

In this scene, a green Ford Mustang is made to run into the ego vehicle from across the intersection.

This scenario is briefly demonstrated in the image sequence below.

Figure 50: Test Scenario 4, Image 1

Figure 51: Test Scenario 4, Image 2

Object	Location	Direction	State w.r.t ego vehicle	Time to Collision
Ego Vehicle	**(-41.2, 128.7)**	**178 SE**	**NA**	**NA**
Ninja bike	**(-73.7, 128.7)**	**88 NE**	**Caution**	**NA**
Green car	**(-87.6, 136.2)**	**10 NW**	**Caution**	**1.3s**

Table 12: Raw Data - test scenario 4, image 2

Figure 52: Test Scenario 4, Image 3

Object	Location	Direction	State w.r.t ego vehicle	Time to Collision
Ego Vehicle	**(-49.8, 128.7)**	**178 SE**	**NA**	**NA**
Ninja bike	**(-73.9, 140.4)**	**88 NE**	**Caution**	**NA**
Green car	**(-81.2, 135.4)**	**-13 NW**	**Danger**	**0.7s**

Table 13: Raw Data - test scenario 4, image 3

Figure 53: Test Scenario 4, Image 4

Object	Location	Direction	State w.r.t ego vehicle	Time to Collision
Ego Vehicle	**(-60.9, 128.7)**	**178 SE**	**NA**	**NA**
Ninja bike	**NA**	**NA**	**NA**	**NA**
Green car	**(-70.1, 132.1)**	**-16 NW**	**Danger**	**0.1s**

Table 14: Raw Data - test scenario 4, image 4

4.3.5 Test Scenario 5: A car taking unethical turn and colliding into the ego vehicle

This test case has been implemented from the green car's point of view. This is just to demonstrate the flexibility of the proposed scenario generation tool such that the same scenario can be run and

observed from so many different points of view. This means more data collection and more scenarios to test on. This scenario is briefly demonstrated in the image sequence below.

Figure 54: Test Scenario 5, Image 1

Figure 55: Test Scenario 5, Image 2

Object	Location	Direction	State w.r.t ego vehicle	Time to Collision
Ego Vehicle	**(-98.5, 139.3)**	**-3 NW**	**NA**	**NA**
Light Blue car	**(-88.5, 145.3)**	**-115 SW**	**Danger**	**0.3s**
Ninja bike	**(-73.9, 124.3)**	**88 NE**	**Caution**	**NA**

Table 15: Raw Data - test scenario 5, image 2

Figure 56: Test Scenario 5, Image 3

Object	Location	Direction	State w.r.t ego vehicle	Time to Collision
Ego Vehicle	**(-98.5, 139.3)**	**-3 NW**	**NA**	**NA**
Light Blue car	**(-91.4, 141.9)**	**-135 SW**	**Danger**	**0.1s**
Ninja bike	**(-73.9, 127.3)**	**88 NE**	**Caution**	**NA**

Table 16: Raw Data - test scenario 5, image 3

4.4 Results Comparison and Discussion

4.4.1 Advantages of Proposed Approach

The main advantage of the proposed approach is the ease and flexibility at which regular and test driving scenarios can be generated. Moreover, the use of highly photo-realistic and high fidelity physics modules helps in closing the reality gap that has always been a concern while using simulation environments. In the past, open-source simulation environments have been widely used for task-specific causes, such as lane detection, testing braking system, etc. The simulation environments used for such causes are usually primitive and treat the traffic objects as mere point masses or 3D boxes at best. On the other hand, our test scenario generation tool is built upon a high fidelity driving simulator, CARLA. This enables the collection of highly realistic data that can be further used for training and testing Autonomous Vehicle algorithms.

We can directly compare our work with a similar data collection tool given in [70]. They have used the same simulator as us, i.e. CARLA, but the scene generation approach is quite different. In their approach, they spawn a lot of actors at designated spawn points and those actors follow their natural movement patterns on an Autopilot mode. Due to this reason, the scenes and data generated is generic and hardly involve any complicated test scenarios that we are looking for. On the other hand, the tool built in the work allows the tester to customize all the details for the scene built up and hence allows to explicitly define the scenario in a flexible manner.

There is another similar scenario and data generation approach implemented using the Autono Vi-Sim simulator by Best et. al in [12]. While the scenarios and data generated by them are similar in nature to our work, the simulator used by them is not highly photo-realistic and also lacks high-fidelity sensors. This makes the data generated by them less close to reality as compared to our

data. In this manner, the algorithms can be more efficiently trained and tested on our data as compared to that in [12]. Here, we haven't conducted any experiments on the data generated in [12] to obtain any statistics as it was out of the scope of our work. However, in their work they have themselves claimed that the tool they used in still in active development and needs improvement to its physics engine as well as sensor modules (lack of photorealism of the camera sensors is clearly evident in their data).The below image sequence demonstrates the difference between the two works in terms of photo-realism and traffic scene features.

Figure 57: Data generated in [12]

Figure 58: Data generated in this thesis

We have already discussed the drawbacks of existing traditional datasets in section 2.2.2. The information about how the data generated by our scenario generation tool covers those drawbacks is shown in Table 17 below. Also, Table 1 in section 2.4.4 can be referred to observe how other simulation environments cover those drawbacks.

Geographical Diversity	Selection Bias	Negative Bias	Capture Bias	Challenging weather and lighting conditions
No	Not Applicable	Yes	Yes	Yes
This can be fulfilled once custom 3D landscape models can be imported and used.		Negative data samples can be collected by arranging the desired objects in desired positions and making them move in a desired manner. (Here Negative means safe and ordinary scenarios where the Neural Network model doesn't have much to learn).	Data can be captured from multiple points of view by attaching sensors to as many objects as required, depending on the processing power of the computer (GPU only) used.	A variety of weather and lighting conditions can be defined and used as appropriate.

Table 17: Drawbacks covered by our data.

4.4.2 Limitations of the Proposed Approach

The main limitation of this test scenario generation tool is that we have to manage with the landscapes provided by the simulator used. We can be hopeful that CARLA, in its further stable releases, allows usage of custom-built .fbx or .obj 3D environments and at the same time, retaining all the functionalities provided by the simulator. Another driving simulator, Microsoft AirSim offers the option of using custom-built environments, but it lacks in terms of user-friendliness, and

also as is not as mature as CARLA. In general, it is a fact that open-source driving simulators are still in their infancy, and their effective usage for Autonomous Vehicles research is gaining momentum at a slow pace.

Chapter 5: Conclusion and Future Work

5.1 Conclusion

Based on the work done in this thesis, we conclude that we have presented a Simulation Environment with Reduced Reality Gap for data generation and testing Autonomous vehicle algorithms in a safe, fast, and cost-effective manner. We have provided a proof of concept of how real traffic scenarios can be replicated quickly in the simulation environment. We further demonstrated how those base scenarios can be edited and various complex test scenarios can be generated using the simulation environment set up in this work. It is safe to say that such developments in Computer Vision would be crucial steps towards fulfilling the qualitative and quantitive data requirements faced by Machine Learning algorithms. This would, in turn, solidify the Testing and Validation of Autonomous Vehicles in myriad scenarios, enabling the Autonomous Vehicles to learn more and get mature enough to replace human drivers and hence, change transportation forever.

5.2 Future Work

This section discusses the future work that can be carried out in this direction.

- As soon as the driving simulators get mature enough to allow the importing and usage of custom-built 3D object models while retaining all the functionalities, the landscapes, and other object models, the models used in this simulation environment can be replaced with actual custom ones, which would be a huge leap towards reducing the Reality gap.
- This simulation environment can be further used for Reinforcement Learning, which is usually the main purpose of driving simulators.

- This simulation environment can be customized further to facilitate more impressive Data Visualization.
- Higher levels of Domain Randomization can be practiced by including non-relevant object models in the traffic scenes.
- This simulation environment can be used to validate other Autonomous Vehicle algorithms like Object Recognition, Object Tracking, Semantic Segmentation, Scene Understanding, etc.
- Apart from the data generated in this work, this simulation environment can be used to implement a huge number of different test scenarios and generate large volumes of complex and realistic data (photographic).
- In this work, we have generated and analyzed just the photographic data. However, while implementing further test scenarios, sensors like radar and Lidar can be easily activated and 3D point cloud data can be generated.

References

[1] "Self-driving Car Market Global Industry Trends, Share, Size and Forecast Report By 2023|With CAGR of 36.2%," Market Watch, 3 September 2019. [Online]. Available: https://www.marketwatch.com/press-release/self-driving-car-market-global-industry-trends-share-size-and-forecast-report-by-2023with-cagr-of-362-2019-09-03. [Accessed 16 November 2019].

[2] R. Spence, "Everything you need to know about the future of self-driving cars," Maclean's, 11 July 2019. [Online]. Available: https://www.macleans.ca/society/technology/the-future-of-self-driving-cars/. [Accessed 16 November 2019].

[3] Daya Driving School, [Online]. Available: pinterest.ca/pin/716564990684968118/?autologin=true. [Accessed 17 November 2019].

[4] A. Acharya, 3 January 2017. [Online]. Available: https://atul.fyi/post/2017/01/03/how-self-driving-cars-work/. [Accessed 16 November 2019].

[5] C. Badue, . R. Guidolini, R. V. Carneiro, P. Azevedo, V. B. Cardoso, A. Forechi, L. Jesus, R. Berriel, T. Paixao and F. Mutz, "Self-Driving Cars: A Survey," in *arXiv:1901.04407v2*, 2019.

[6] J. Koh, "Object detection with LiDAR Point cloud Algorithm," Meduim, 1 November 2018. [Online]. Available: https://medium.com/@jhkoh/object-detection-with-lidar-point-cloud-algorithm-94a241fd3f49. [Accessed 17 November 2019].

[7] J. Kocić, N. Jovičić and V. Drndarević, "Sensors and Sensor Fusion in Autonomous," in *2018 26th Telecommunications Forum (TELFOR), Belgrade, Serbia, 20–21 November 2018; pp. 420–425*, Belgrade, 2018.

[8] D. Silver, "Self-Driving Path Planning, Brought to You by Udacity Students," Udacity, 25 August 2017. [Online]. Available: https://medium.com/udacity/self-driving-path-planning-brought-to-you-by-udacity-students-13c07bcd4f32. [Accessed 17 November 2019].

[9] M. O'Kelly, A. Sinha and H. Namkoong, "Scalable End-to-End Autonomous Vehicle," in *32nd Conference on Neural Information Processing Systems (NeurIPS 2018)*, Montreal, 2018.

[10] N. Kalra and S. Paddock, "Driving to Safety: How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability?," in *https://www.rand.org/pubs/research_reports/RR1478.html*, Santa Monica, 2016.

[11] "Fatal Uber crash shows risks of testing on public roads," IIHS, 7 August 2018. [Online]. Available: https://www.iihs.org/news/detail/fatal-uber-crash-shows-risks-of-testing-on-public-roads. [Accessed 17 November 2019].

[12] A. Best, S. Narang, L. Pasqualin, D. Barber and D. Manocha, "AutonoVi-Sim: Autonomous Vehicle Simulation Platform with Weather, Sensing,," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops. 2018.*, 2018.

[13] F. Rosique, P. Navarro, C. Fernández and A. Padilla, "A Systematic Review of Perception System and," in *mdpi*, 2019.

[14] "Simulations Pave the Road for Self-Driving Technologies," Synced, 8 April 2018. [Online]. Available: https://medium.com/syncedreview/simulations-pave-the-road-for-self-driving-technologies-78b696227383. [Accessed 17 November 2019].

[15] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection," in *arXiv:1506.01497v3*, 2015.

[16] R. Girshick, J. Donahue, T. Darrell and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *arXiv:1311.2524v5*, 2014.

[17] R. Girshick, "Fast R-CNN," in *arXiv:1504.08083*, 2015.

[18] S. Goswami, "A deeper look at how Faster-RCNN works," Medium, 11 July 2018. [Online]. Available: https://medium.com/@whatdhack/a-deeper-look-at-how-faster-rcnn-works-84081284e1cd. [Accessed 18 November 2019].

[19] K. Simonyan and A. Zisserman, "VERY DEEP CONVOLUTIONAL," in *arXiv:1409.1556v6*, 2015.

[20] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once:," in *arXiv:1506.02640v5 [cs.CV]*, 2015.

[21] A. Torralba and A. Efros, "Unbiased Look at Dataset Bias," in *CVPR 2011*, Providence, 2011.

[22] A. Geiger, P. Lenz, C. Stiller and R. Urtasun, "Vision meets Robotics: The KITTI Dataset," 2013.

[23] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth and B. Schiele, "The Cityscapes Dataset for Semantic Urban Scene Understanding," in *arXiv:1604.01685v2 [cs.CV]*, 2016.

[24] "ImageNet," ImageNet Project, [Online]. Available: http://www.image-net.org/. [Accessed 19 November 2019].

[25] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, . D. Ramanan, C. L. Zitnick and P. Dollar, "Microsoft COCO: Common Objects in Context," in *arXiv:1405.0312v3 [cs.CV]*, 2015.

[26] H. Rowley, S. Baluja and t. Kanade, "Neural Network-Based Face Detection," in *IEEE*, 1998.

[27] W. Maddern, G. Pascoe, C. Linegar and P. Newman, "1 Year, 1000km: The Oxford RobotCar Dataset," in *Data Papers*, 2016.

[28] D. J. Butler, J. Wulff, G. B. Stanley and M. J. Black, "A Naturalistic Open Source Movie," in *Springer-Verlag Berlin Heidelberg*, 2015.

[29] A. Dosovitski, P. Fischer, E. Ilg, . P. Hausser, C. Hazırbas, V. Golkov, P. van der Smagt, D. Cremers and T. Brox, "FlowNet: Learning Optical Flow with Convolutional Networks," in *arXiv:1504.06852v2 [cs.CV]* , 2015.

[30] A. Handa, V. Patr ˇ aucean, V. Badrinarayanan, S. Stent and R. Cipolla, "SceneNet: Understanding Real World Indoor Scenes With Synthetic Data," in *arXiv:1511.07041v2 [cs.CV]* , 2015.

[31] G. Ros, L. Sellart, J. Materzynska, D. Vazquez and A. M. Lopez, "The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic," in *IEEE*, Las Vegas, 2016.

[32] "Make Something Unreal with the most powerful creation engine," [Online]. Available: https://www.unrealengine.com/en-US/what-is-unreal-engine-4. [Accessed 20 November 2019].

[33] "Unity for all," [Online]. Available: https://unity.com/. [Accessed 20 November 2019].

[34] "Blender," [Online]. Available: https://www.blender.org/. [Accessed 20 November 2019].

[35] "CRYENGINE," [Online]. Available: https://www.cryengine.com/. [Accessed 20 November 2019].

[36] S. Shah, D. Dey, C. Lovett and A. Kapoor, "AirSim: High-Fidelity Visual and Physical," in *Springer International Publishing*, 2018.

[37] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez and V. Koltun, "CARLA: An Open Urban Driving Simulator," in *arXiv:1711.03938v1 [cs.LG]*, 2017.

[38] "Getting started with CARLA," CARLA , [Online]. Available: https://carla.readthedocs.io/en/latest/getting_started/. [Accessed 2019 November 21].

[39] D. Dworak, F. Ciepiela, J. Derbisz, I. Izzat, M. Komorkiewicz and M. Wojcik, "Performance of LiDAR object detection deep learning architectures based on artificially generated point cloud data from CARLA simulator," in *2019 24th International Conference on Methods and Models in Automation and Robotics (MMAR)*, Międzyzdroje, 2019.

[40] Y. Jaafra, J. L. Laurent, A. Deruyver and M. S. Naceur, "Seeking for Robustness in Reinforcement Learning: Application on Carla," in *International Conference on Machine Learning (ICML)*, 2019.

[41] F. Yang, P. Wang and X. Wang, "Continuous Control in Car Simulator with Deep Reinforcement Learning," in *CSAI '18 Proceedings of the 2018 2nd International Conference on Computer Science and Artificial Intelligence Pages 566-570*, Shenzen, 2018.

[42] M. Jasinski, "A Generic Validation Scheme for real-time capable Automotive Radar Sensor Models integrated into an Autonomous Driving Simulator," in *2019 24th International Conference on Methods and Models in Automation and Robotics (MMAR)*, Międzyzdroje, 2019.

[43] Q. Chao, X. Jin, H.-W. Huang, S. Foong, L.-F. Yu and S.-K. Yeung, "Force-based Heterogeneous Traffic Simulation for Autonomous Vehicle Testing," in *2019 International Conference on Robotics and Automation (ICRA)*, Montreal, 2019.

[44] K. Srivastava, A. K. Singh and G. M. Hegde, "Multi Modal Semantic Segmentation using Synthetic Data," in *Deep Learning for Automated Driving (DLAD) workshop, IEEE International Conference on Intelligent Transportation Systems (ITSC'19)*, 2019.

[45] "AirSim Home," Microsoft, [Online]. Available: https://microsoft.github.io/AirSim/. [Accessed 21 November 2019].

[46] A. Kapoor and S. Shah, "Microsoft AirSim now available on Unity," Microsoft, 14 November 2018. [Online]. Available: https://www.microsoft.com/en-us/research/blog/microsoft-airsim-now-available-on-unity/. [Accessed 21 November 2019].

[47] "DeepDrive SImulator," DeepDrive, [Online]. Available: https://deepdrive.io/index.html. [Accessed 22 November 2019].

[48] "udacity/self-driving-car-sim," Udacity, [Online]. Available: https://github.com/udacity/self-driving-car-sim. [Accessed 22 November 2019].

[49] "NVIDIA DRIVE CONSTELLATION Virtual Reality Autonomous Vehicle Simulator," [Online]. Available: https://www.nvidia.com/en-us/self-driving-cars/drive-constellation/. [Accessed 22 November 2019].

[50] "Waymo details ‘Carcraft’ simulation software, ‘Castle’ testing site for self-driving car training," 9to5google, [Online]. Available: https://ww.9to5google.com/2017/08/23/waymo-self-driving-carcraft-software-castle-testing/. [Accessed 22 November 2019].

[51] V. Me´ndez, H. Catala´n, J. R. Rosell, J. Arno´, R. Sanz and A. Tarquis, "SIMLIDAReSimulation of LIDAR performance in artificiallysimulated orchards," 2011.

[52] "GIScience/helios," [Online]. Available: https://github.com/GIScience/helios. [Accessed 22 November 2019].

[53] "RasSimCT," [Online]. Available: https://radsimct.se/. [Accessed 22 November 2019].

[54] "SimSonic," SimSonic Development, [Online]. Available: http://www.simsonic.fr/.

[55] K. Bousmalis and S. Levine, "Closing the Simulation-to-Reality Gap for Deep Robotic Learning," googleai, 30 October 2017. [Online]. Available: https://ai.googleblog.com/2017/10/closing-simulation-to-reality-gap-for.html. [Accessed 22 November 2019].

[56] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba and P. Abbeel, "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2017)*, 2017.

[57] J. Borrego, A. Dehban, R. Figueiredo, P. Moreno, A. Bernardino and . J. Santos-Victor, "Applying Domain Randomization to Synthetic Data for Object Category Detection," in *arXiv:1807.09834v1 [cs.CV]*, 2018.

[58] T. Tremblay, A. Prakash, D. Acuna, M. Brophy, V. Jampani, A. Cem, T. To, E. Cameracci, S. Cameracci and S. Birchfield, "Training Deep Networks with Synthetic Data: Bridging the Reality Gap by Domain Randomization," in *CVPR 2018 Workshop on Autonomous Driving*, 2018.

[59] X. Yue, Y. Zhang, S. Zhao, A. Sangiovanni-Vincentelli, K. Keutzer and B. Gong, "Domain Randomization and Pyramid Consistency: Simulation-to-Real Generalization without Accessing Target Domain Data," in *arXiv:1909.00889v1 [cs.CV]*, 2019.

[60] A. Prakash, S. Boochoon, M. Brophy, D. Acuna, E. Cameracci, G. State, O. Shapira and S. Birchfield, "Structured Domain Randomization: Bridging the Reality Gap by Context-Aware Synthetic Data," in *arXiv:1810.10093v1 [cs.CV]*, 2018.

[61] M. Johnson-Roberson, C. Barto, R. Mehta, S. N. Sridhar, K. Rosaen and R. Vasudevan, "Driving in the Matrix: Can Virtual Worlds Replace Human-Generated Annotations for Real World Tasks?," in *arXiv:1610.01983v2 [cs.CV]*, 2017.

[62] "List of self-driving car fatalities," Wikipedia, 15 October 2019. [Online]. Available: https://en.wikipedia.org/wiki/List_of_self-driving_car_fatalities. [Accessed 26 November 2019].

[63] F.-H. Chan, Y.-T. Chen, Y. Xiang and M. Sun, "Anticipating Accidents in Dashcam Videos," in *ACCV: Asian Conference on Computer Vision*, Taipei, 2016.

[64] "smallcorgi/Anticipating-Accidents," 2016. [Online]. Available: https://github.com/smallcorgi/Anticipating-Accidents. [Accessed 22 October 2019].

[65] Y. Wang and J. Kato, "Collision Risk Rating of Traffic Scene from," in *International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, Sydney, 2017.

[66] H. Kim, K. Lee, G. Hwang and C. Suh, "Crash to Not Crash: Learn to Identify Dangerous Vehicles using a Simulator," in *AAAI Conference on Artificial Intelligence*, 2019.

[67] P. Feth, M. N. Akran, R. Schuster and O. Wasenm¨uller, "Dynamic Risk Assessment for Vehicles," in *International Workshop on Artificial Intelligence Safety Engineering (WAISE)*, 2018.

[68] D. Phillips, J. C. Aragon, A. Roychowdhury, R. Madigan, S. Chintakindi and M. Kochenderfer, "Real-time Prediction of Automotive Collision Risk," in *arXiv:1902.01293v1 [cs.CV]*, 2019.

[69] G. Corcoran and J. Clark, "Traffic Risk Assessment: A Two-Stream Approach Using Dynamic-Attention," in *16th Conference on Computer and Robot Vision (CRV)*, 2019.

[70] "Github," [Online]. Available: https://github.com/carla-simulator/data-collector. [Accessed 06 12 2019].

Vita Auctoris

Name: Kaival Kamleshkumar Patel

Birth Place: Ahmedabad, Gujarat, India

Birth Year: 1995

Education: Bachelor of Engineering (B.E.), 2012-2016

Information Technology (IT)

L. D. College of Engineering, Ahmedabad, Gujarat, India

(affiliated under *Gujarat Technological University*, Gujarat, India (GTU))

Master of Science (MSc.) with Co-op, Fall 2017-Fall 2019

Computer Science (CS)

School of Computer Science

University of Windsor, Windsor, Ontario, Canada (UoW)